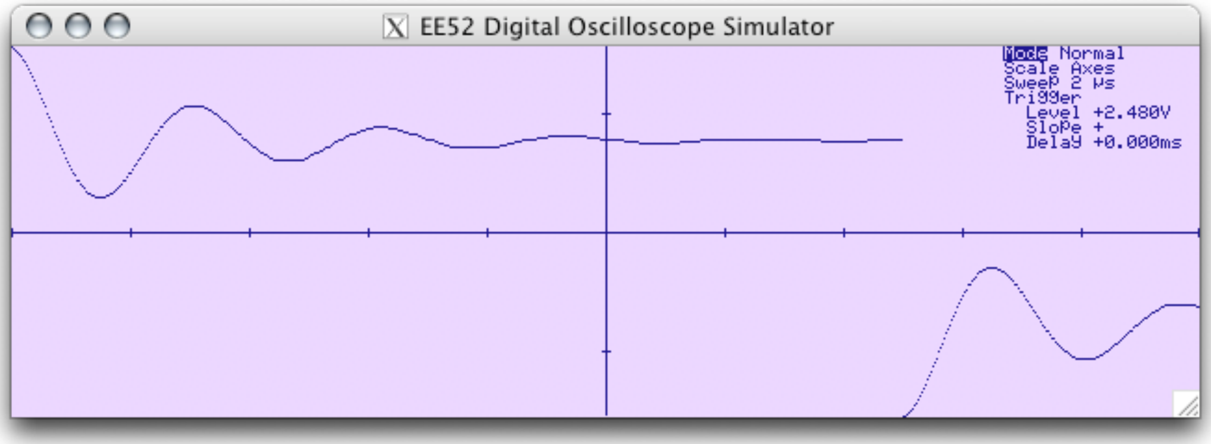
EE/CS 52 Digital Oscilloscope Documentation

Julian Panetta
TA: Gabe Cohn

Table of Contents

Digital Oscilloscope

User's Manual



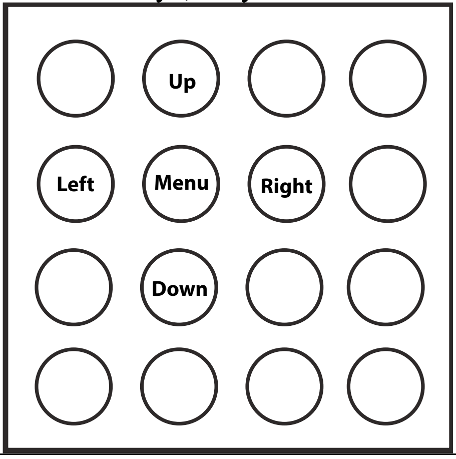
```
Mode Normal
Scale Axes
Sweep 2 μs
Trigger
  Level +2.480V
  Slope +
  Delay +0.000ms
```

Introduction

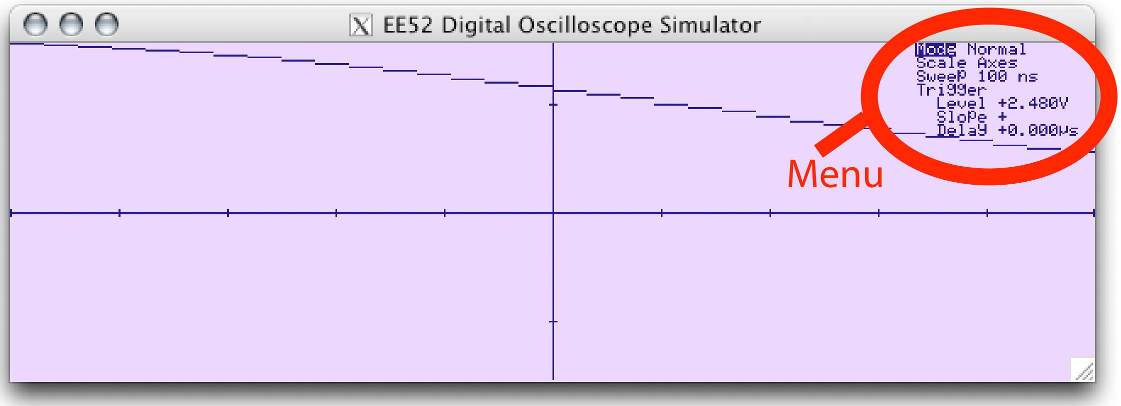
This digital oscilloscope is an easy-to-use instrument for visualizing analog input signals. In fact, such great care has been taken to ensure ease of use that the user's task of finding and attaching an analog voltage signal source has been removed entirely. Simply attach the oscilloscope to a computer, open up your $2500+ copy of Quartus, program the FPGA design into it, and you're ready to view the beautiful, noise-free built-in simulated signal. Even better, there is no need to try to figure out how switch to another input signal because this oscilloscope has been specialized to display only this signal!

Hardware

The hardware devices you use to interact with and monitor the oscilloscope are described on the next page.

Device Name	Device Type	Description
Keypad	4*4 Array of 16 keys	Of the 16 keys, only 5 are used: 
Display	LCD Panel	A “high” quality LCD to display the oscilloscope’s output.
Status LED	LED	A shiny red LED that lights when system has booted to let you know that initialization has completed successfully and the scope interface should be running.

User Interface



```
Mode Normal
Scale Axes
Sweep 100 ns
Trigger
  Level +2.480V
  Slope +
  Delay +0.000μs
```

Once programmed, the system boots into Normal mode with a 100ns sampling rate and a positive slope mid-level (2.480V) trigger with no delay. x- and y-axes are displayed behind the trace, and a menu in the upper right-hand corner—demonstrated in the screenshot above—lets you change all of the scope’s settings and enter different sampling modes. To hide and show the menu, press the <Menu> key.

You can change from one highlighted menu item to the next using the <Up> and <Down> keys. To alter the highlighted configuration setting, the user presses the <Left> and <Right> keys to select a value.

The configurable settings and their values are as follows:

Menu Title	Options and Description	
Mode	Normal	Scope waits for a trigger after completing each retrace.
	Automatic	Scope waits for a trigger after each retrace, but retriggers automatically without a trigger event after a delay.
	One-Shot	Scope triggers only once, continuing to display the last trace captured.
Scale	Scale Axes	Display x and y axes along with trace.
	Scale Grid	Display x-y grid along with the trace.
	Scale Off	Display the trace only.
Sweep	Items are time quantities that specify the sweep rate, or the time between successive samples options: 100ns, 200ns, 500ns, 1us, 2us, 5us, 10us, 20us, 50us, 100us, 200us, 500us, 1ms, 2ms, 5ms, 10ms, 20ms	
Level	128 voltage quantities are given as choices for the trigger voltage level. The levels are equally spaced from 0V to 5V.	
Slope	Slope +	Scope is triggered on a positive slope (rising edge)
	Slope -	Scope is triggered on a negative slope (falling edge)
Delay	Time quantities ranging between 0ms and 1ms specify the time between the trigger and the start of the trace.	

Limitations

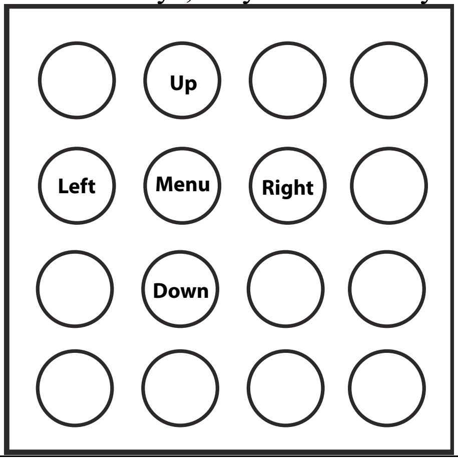
If for some reason you have decided to go through the effort of finding a voltage source you want to measure, you will find that the oscilloscope has no connector to which you can attach the signal. The oscilloscope is compatible only with the built-in demonstration trace.

Digital Oscilloscope

Functional Specification

Description: The system is an interactive digital oscilloscope displaying on a 640 x 200 LCD panel. The signal is input via an analog line in, which runs through an analog to digital converter. There is a keypad to change the operation parameters. The system must be connected to a computer to program the FPGA, but can be disconnected once booted.

Inputs: Input is received through a keypad, a serial connection, and an analog input signal.

Input Name	Input Type	Description
Keypad	4*4 Array of 16 keys	Of the 16 keys, only 5 are used by the OS: 
Serial	RS-232 Port	Allows communication with the NIOS using GERMS. Downloading of sample data to the oscilloscope from the computer is not yet implemented.
Analog	Analog Line In	Analog signal line whose voltage variation with time the scope measures. This portion of the oscilloscope is not yet implemented.

Outputs: Output is implemented with a display, a serial connection, and a LED.

Output Name	Output Type	Description
Display	LCD Panel	A 640 x 200 graphics panel used to display the analog input and to present an operation and configuration menu to the user.
Serial	RS-232 Port	Allows sampled data to be uploaded to a

		computer via a serial connection. This feature is not yet implemented.
Status LED	LED	LED lights when system has booted to inform user that the software initialization has completed successfully and the OS should be running.

User Interface: By default, the system boots into Normal mode with a 100ns sampling rate and a positive slope mid-level (2.480V) trigger with no delay. x- and y-axes are displayed behind the trace, and a menu in the upper right-hand corner lets the user change all of the scope's settings and enter different sampling modes. The user can press the menu key to toggle the menu's visibility.

The user changes from one highlighted menu item to the next using the up and down keys. To alter the highlighted configuration setting, the user presses the left and right keys to select a value.

Serial transfers will be implemented with "silent linking." The computer will send a request to transmit data or a request to receive data, and the oscilloscope will comply without asking the user to confirm. A status message for the transfer will be displayed in the lower left-hand corner.

The menu options are as follows:

Menu Title	Options and Description	
Mode	Normal	Scope waits for a trigger after completing each retrace.
	Automatic	Scope waits for a trigger after each retrace, but retriggers automatically without a trigger event after a delay.
	One-Shot	Scope triggers only once, continuing to display the last trace captured.
Scale	Scale Axes	Display x and y axes along with trace.
	Scale Grid	Display x-y grid along with the trace.
	Scale Off	Display the trace only.
Sweep	Items are time quantities that specify the sweep rate, or the time between successive samples options: 100ns, 200ns, 500ns, 1us, 2us, 5us, 10us, 20us, 50us, 100us, 200us, 500us, 1ms, 2ms, 5ms, 10ms, 20ms	
Level	128 voltage quantities are given as choices for the trigger voltage level. The levels are equally spaced from 0V to 5V.	
Slope	Slope +	Scope is triggered on a positive slope (rising edge)
	Slope -	Scope is triggered on a negative slope (falling edge)
Delay	Time quantities ranging between 0ms and 1ms specify the time between the trigger and the start of the trace.	

Error Handling: If a key not mapped to an interface action in the current mode of operation is pressed, then no operation occurs. No other errors are anticipated for the current implementation.

Once analog and serial input is implemented, the following error handling will apply:

> If a buffer overflow occurs on the signal input queue, “Signal Buffer Overflow” is displayed in the lower left-hand corner, and no further input is accepted. If a serial overflow occurs, “Serial Buffer Overflow” is displayed. If an error with framing, parity, a break, or some other serial error occurs, the messages “Framing Error”, “Parity Error”, “Break Error”, and “Serial Error” are displayed respectively.

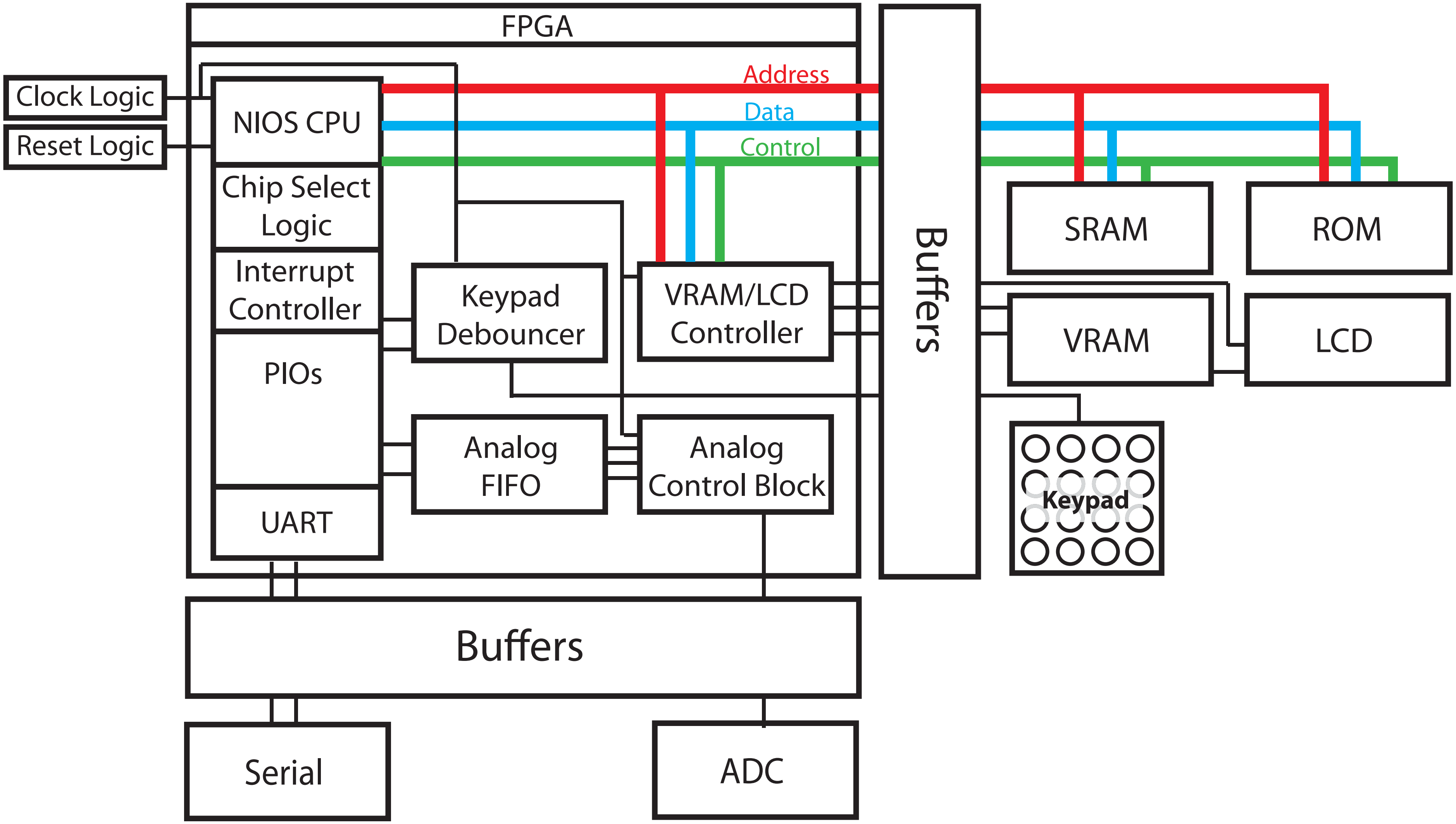
Algorithms: A FFT may be implemented in the future for data analysis purposes. Currently no FFT is used.

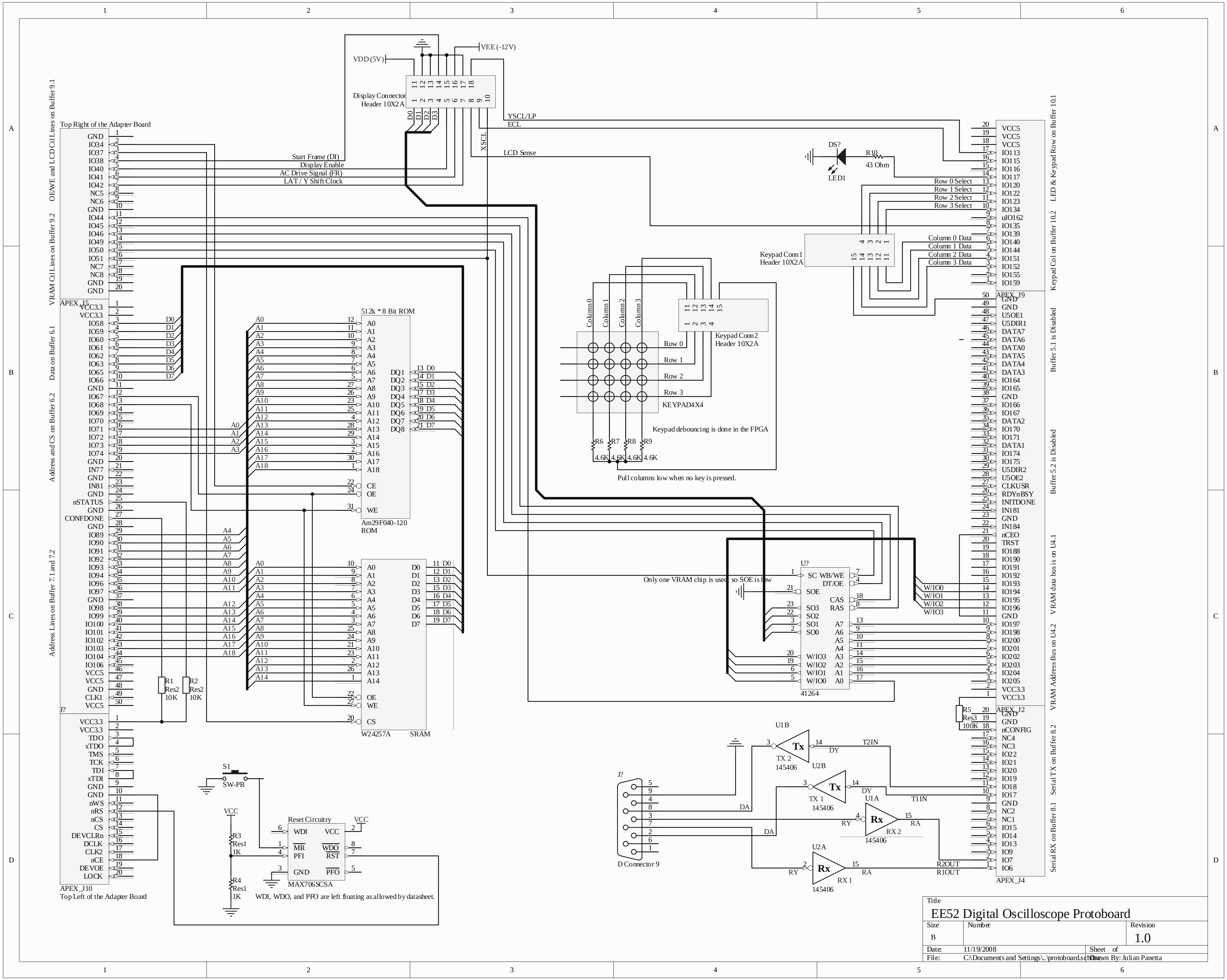
A look up table is used to validate the user’s keypresses.

Data Structures: A circular queue will be used to buffer serial input in case bytes are received faster than the NIOS can respond. A FIFO will also be used to buffer the signal input from the analog to digital converter/analog control block.

Limitations: There is currently no analog or serial input.

EE/CS 52 Digital Oscilloscope Block Diagram





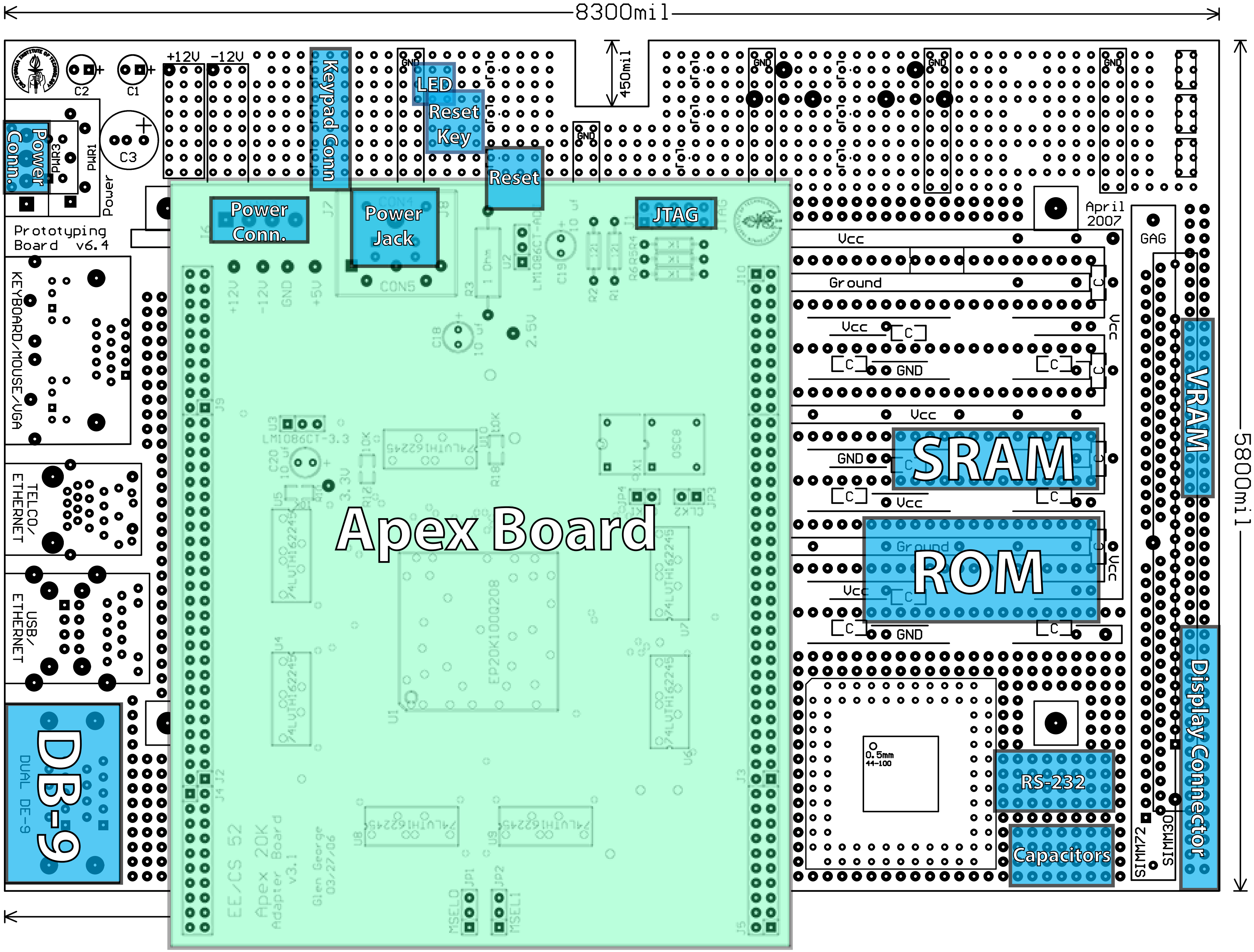
Title

EE52 Digital Oscilloscope Protoboard

Size	Number	Revision
B		1.0

Date: 11/19/2008 Sheet of

File: C:\Documents and Settings\..\protoboard.schDoc Drawn By: Julian Panetta


8300mil
450mil
5800mil
+12V
-12V
C1
C2
C3
PWR1
PWR3
Power
Power Conn.
Prototyping Board v6.4
Keypad Conn
LED
Reset Key
Reset
GND
Power Conn.
Power Jack
JTAG
April 2007
GAG
Vcc
Ground
GND
KEYBOARD/MOUSE/VGA
TELCO/ETHERNET
USB/ETHERNET
Apex Board
SRAM
ROM
VRAM
Display Connector
RS-232
Capacitors
0.5mm 44-100
SIMM72
SIMM30
DB-9
DUAL DE-9
EE/CS 52
Apex 20K Adapter Board v3.1
Glen George 03/27/06
MSEL0
MSEL1
JP1
JP2

Timing Diagrams

- VRAM
  - Timing Specifications
  - Read, Write, Refresh, Data Transfer
- LCD
  - Timing Specifications (1180F, 1190)
  - LCD Signals
- SRAM
  - Read, Write
- Rom
  - Read

VRAM Timing Specifications

DESCRIPTION	SYMBOI	MIN	MAX
Column address hold time after $\overline{\text{RAS}}$ low	t_{AR}	80ns	
Column address setup time	t_{ASC}	0ns	
Row address setup time	t_{ASR}	0ns	
Access time from $\overline{\text{CAS}}$	t_{CAC}		60ns
Column address hold time	t_{CAH}	20ns	
$\overline{\text{CAS}}$ pulse width	t_{CAS}	60ns	
$\overline{\text{DT}}$ low hold time after $\overline{\text{RAS}}$ low	t_{CDH}	40ns	
$\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh hold time	t_{CHR}	25ns	
$\overline{\text{CAS}}$ precharge time (page cycle only)	t_{CP}	50ns	
$\overline{\text{CAS}}$ precharge time (nonpage cycle)	t_{CPN}	25ns	
$\overline{\text{CAS}}$ high to $\overline{\text{RAS}}$ low precharge time	t_{CRP}	10ns	
$\overline{\text{CAS}}$ hold time	t_{CSH}	120ns	
$\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh setup time	t_{CSR}	10ns	
$\overline{\text{CAS}}$ to $\overline{\text{WE}}$ delay	t_{CWD}	100ns	
Write command to $\overline{\text{CAS}}$ lead time	t_{CWL}	40ns	
Data-in hold time	t_{DH}	35ns	
$\overline{\text{DT}}$ high hold time	t_{DHH}	20ns	
Data-in hold time after $\overline{\text{RAS}}$ low	t_{DHR}	95ns	
$\overline{\text{DT}}$ high setup time	t_{DHS}	0	

VRAM Timing Specifications continued...

DESCRIPTION	SYMBOl	MIN	MAX
$\overline{DT}$ low setup time	t_{DLS}	0	
Data-in setup time	t_{DS}	0	
$\overline{DT}$ high to $\overline{CAS}$ high delay	t_{DTC}	10ns	
$\overline{DT}$ high hold time after $\overline{RAS}$ high	t_{DTH}	20ns	
$\overline{DT}$ high to $\overline{RAS}$ high delay	t_{DTR}	10ns	
$\overline{OE}$ pulse width	t_{OE}	35ns	
Access time from $\overline{OE}$	t_{OEA}		30ns
$\overline{OE}$ to data-in setup delay	t_{OED}	35ns	
$\overline{OE}$ hold time after $\overline{WE}$ low	t_{OEH}	30ns	
$\overline{OE}$ to $\overline{RAS}$ inactive setup time	t_{OES}	10ns	
Output disable time from $\overline{OE}$ high	t_{OEZ}	0	30ns
Output disable time from $\overline{CAS}$ high	t_{OFF}	0	30ns
Page cycle time	t_{PC}	120ns	
Access time from $\overline{RAS}$	t_{RAC}		120ns
Row address hold time	t_{RAH}	15ns	
$\overline{RAS}$ pulse width	t_{RAS}	120ns	10000ns
Random read or write cycle time	t_{RC}	220ns	
$\overline{RAS}$ to $\overline{CAS}$ delay time	t_{RCD}	25ns	60ns
Read command hold time after $\overline{CAS}$ high	t_{RCH}	0	

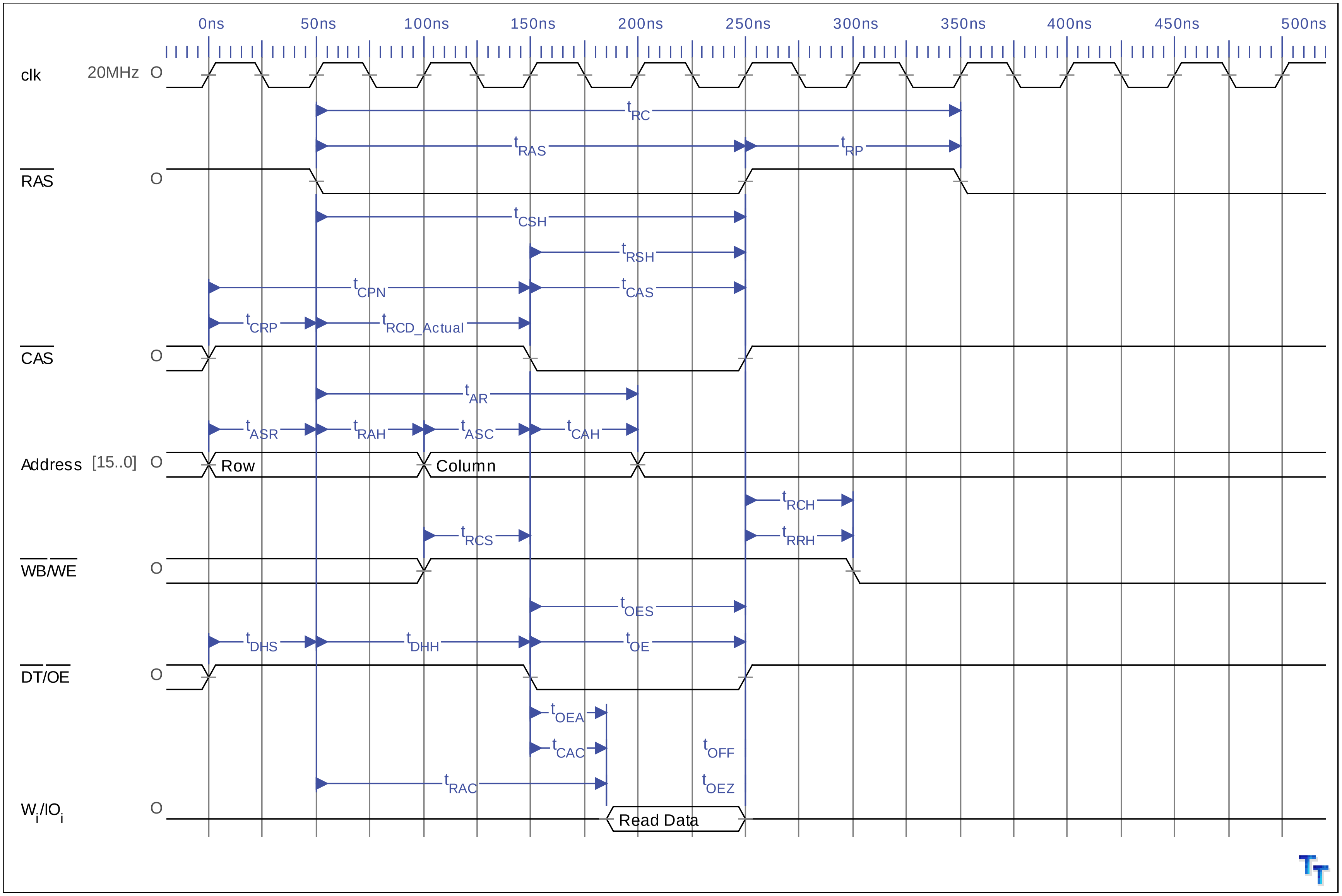
VRAM Timing Specifications continued...

DESCRIPTION	SYMBOL	MIN	MAX
Read command setup time	t_{RCS}	0	
$\overline{DT}$ low hold time after $\overline{RAS}$ low (serial port active)	t_{RDH}	100ns	
Refresh interval	t_{REF}		4ns
$\overline{RAS}$ precharge time	t_{RP}	90ns	
$\overline{RAS}$ high to $\overline{CAS}$ low precharge time	t_{RPC}	0	
Read command hold after $\overline{RAS}$ high	t_{RRH}	20ns	
$\overline{RAS}$ hold time	t_{RSH}	60ns	
Read-write/read-modify-write cycle time	t_{RWC}	300ns	
$\overline{RAS}$ to $\overline{WE}$ delay	t_{RWD}	160ns	
to be defined	t		
Write command to $\overline{RAS}$ lead time	t_{RWL}	40ns	
SC pulse width	t_{SCH}	10ns	
Serial output access time from SC	t_{SCA}		40ns
Serial clock cycle time	t_{SCC}	40ns	50000ns
SC precharge time	t_{SCL}	10ns	
SC high to $\overline{DT}$ high delay	t_{SDD}	10ns	
SC low hold time after $\overline{DT}$ high	t_{SDH}	10ns	
Serial output access time from $\overline{SOE}$	t_{SOA}		35ns
$\overline{SOE}$ pulse width	t_{SOE}	15ns	

VRAM Timing Specifications continued...

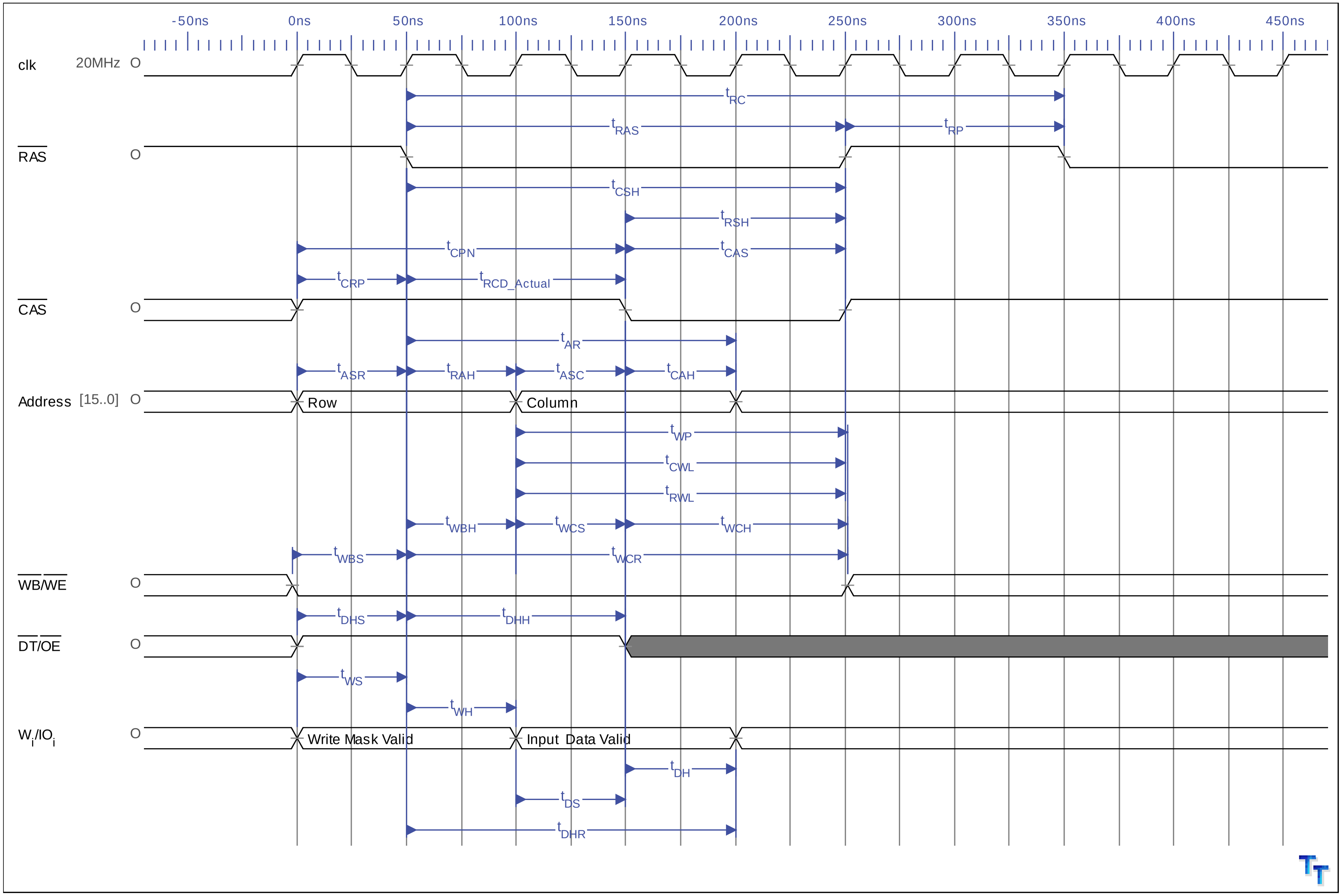
DESCRIPTION	SYMBOL	MIN	MAX
Serial output hold time after SC high	t_{SOH}	10ns	
$\overline{\text{SOE}}$ low to serial output setup delay	t_{SOO}	5ns	
$\overline{\text{SOE}}$ precharge time	t_{SOP}	15ns	
Serial output disable time from $\overline{\text{SOE}}$ high	t_{SOZ}	0	30ns
Rise and fall transition time	t_{T}	3ns	50ns
Write-per-bit hold time	t_{WBH}	20ns	
Write-per-bit setup time	t_{WBS}	0	
Write command hold time	t_{WCH}	35ns	
Write command hold time after $\overline{\text{RAS}}$ low	t_{WCR}	95ns	
Write command setup time	t_{WCS}	0	
Write bit selection hold time	t_{WH}	20ns	
Write command pulse width	t_{WP}	35ns	
Write bit selection setup time	t_{WS}	0	
Actual $\overline{\text{RAS}}$ to $\overline{\text{CAS}}$ delay	t_{RCD_Actual}	100	100

vram_read



C:\Documents and Settings\Administrator\Desktop\vram_read.tdml

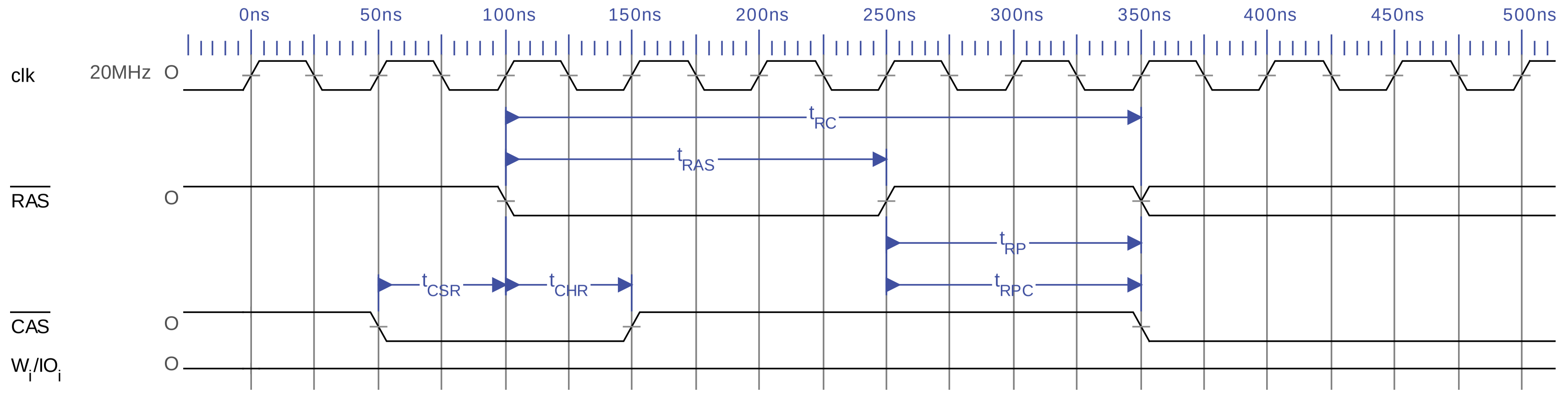
vram_write



C:\Documents and Settings\Administrator\Desktop\vram_write.tdml

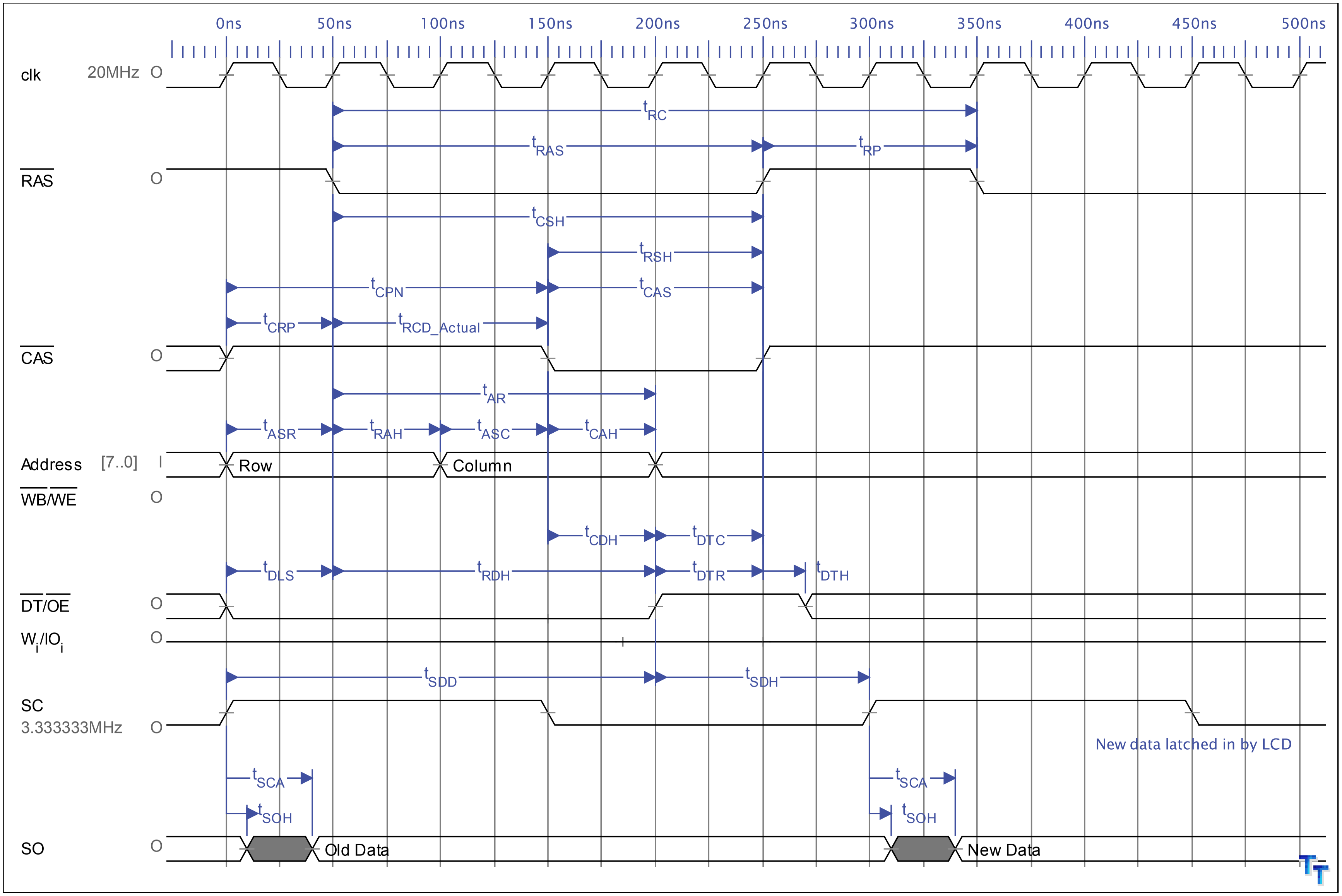
vram_refresh

Page 1



C:\Documents and Settings\Administrator\Desktop\vram_refresh.tdml

VRAM Data Transfer



C:\Documents and Settings\Administrator\Desktop\vram_dt.tdml~

1180F

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
t_{CLC}		shift clock cycle	166		
t_{WCLH}		Shift clock "H" width	63		
t_{WCLL}		Shift clock "L" width	63		
t_{DS}		Data setup time	50		
t_{DH}		Data hold time	30		
t_{WECH}		Enable clock "H" width	100		
t_{WECL}		Enable clock "L" width	100		
t_{EDS}		Enable data setup time	50		
t_{EDH}		Enable data hold time	20		
t_{EDR}		Enable clock delay time	-10		
t_{ECS}		Enable clock setup time	70		
t_{WLPH}		Latch pulse "H" width	110		
t_{WLPL}		Latch pulse "L" width	220		
t_{LT}		Latch timing	100		
t_{LH}		Latch hold time	0		
t_{LDS}		Latch pulse data setup time	140		
t_{LDH}		Latch pulse data hold time	50		
t_{DFR}		Permissible frame signal delay	-500	500	

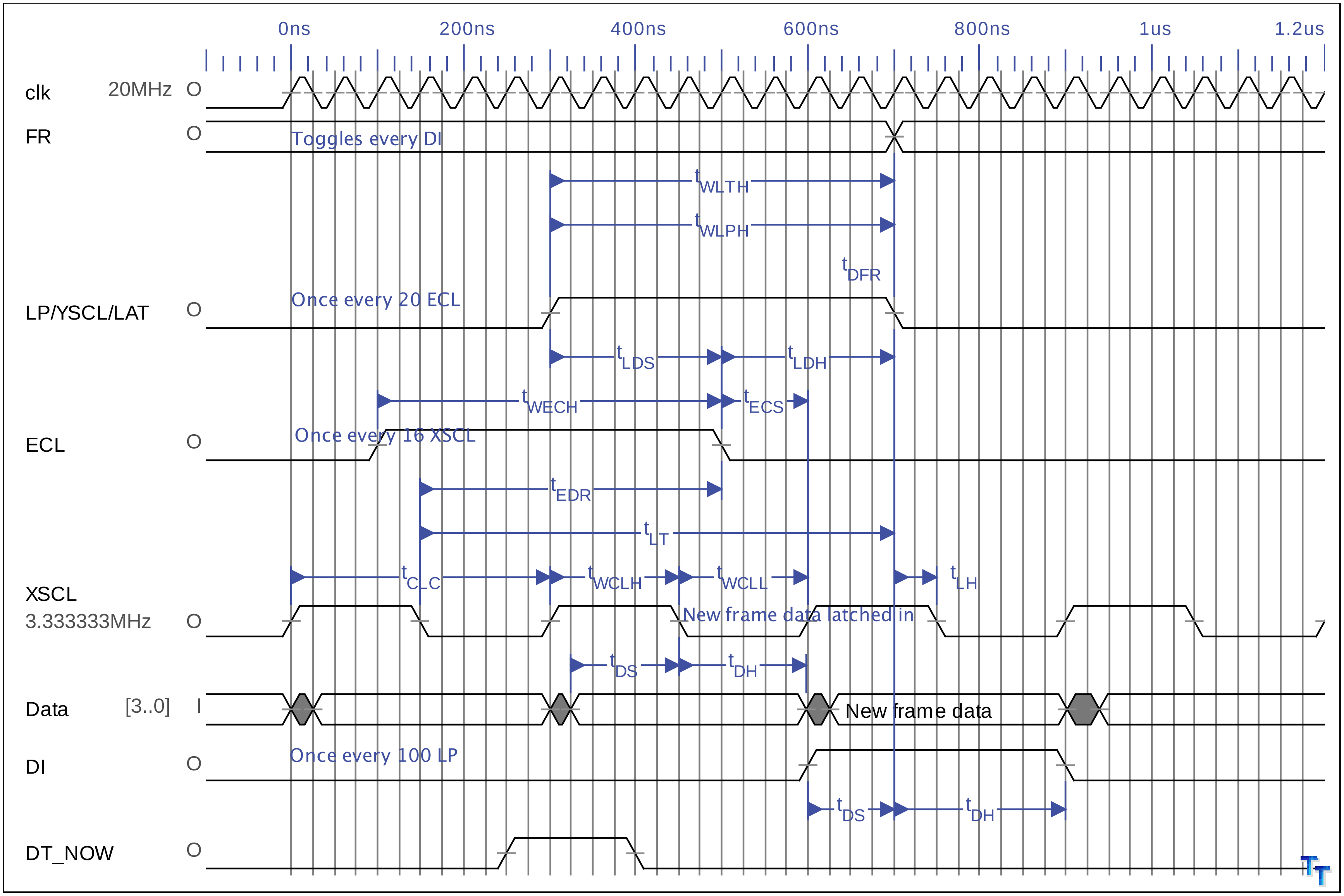
1180F continued...

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
t_{PD}		Enable output delay	20	150	

1190

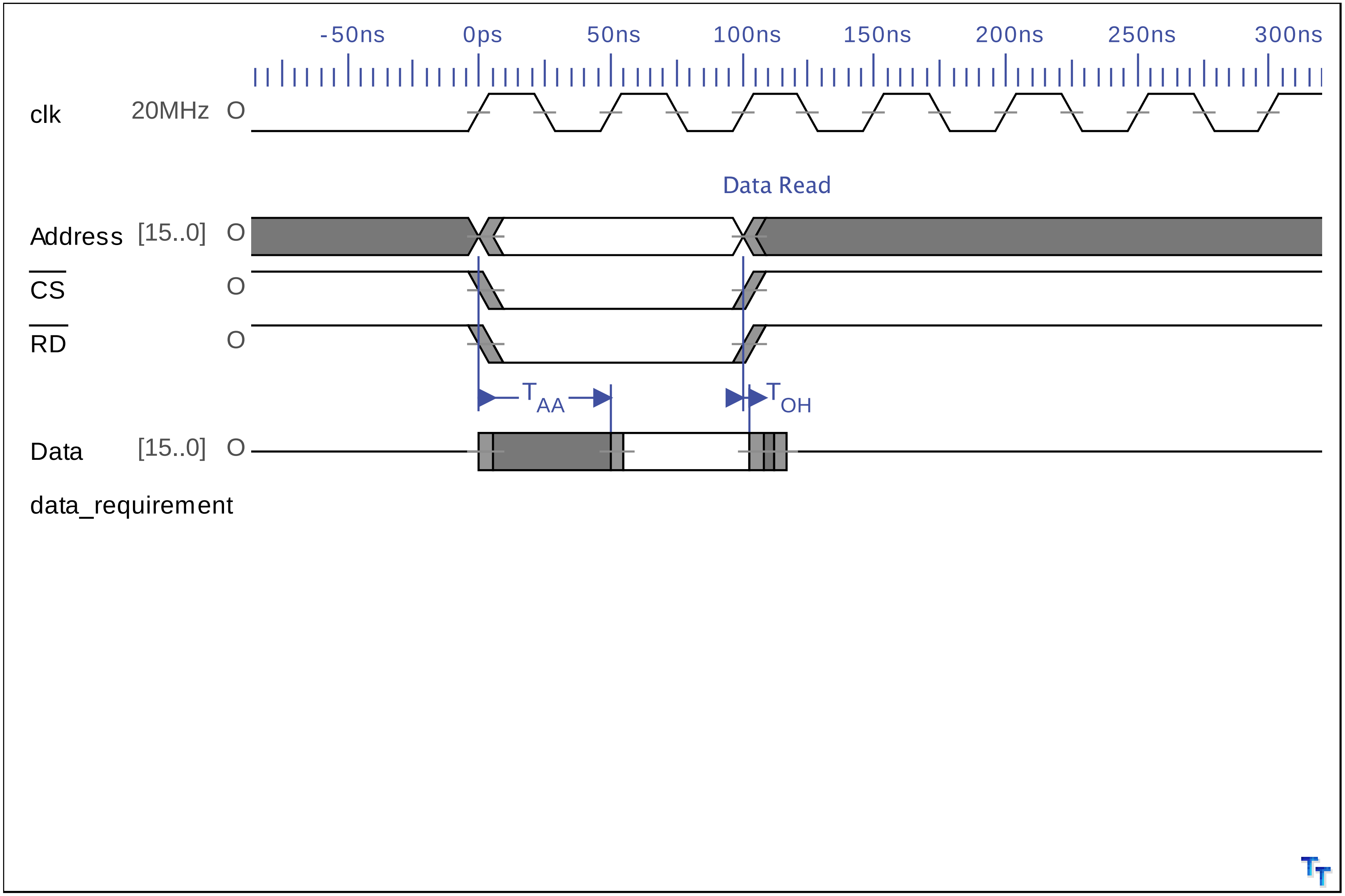
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOTES
t_{CYL}	Latch pulse cycle time	to be defined	400		
t_{WLTH}	Latch pulse "H" width	to be defined	180		
t_{WLL}	Latch pulse "L" width	to be defined	180		
t_{CYC}	Shift clock cycle time	to be defined	400		
t_{WCLH}	Shift clock "H" time	to be defined	110		
t_{WCLL}	Shift clock "L" time	to be defined	240		
t_{DS}	Data setup time	to be defined	70		
t_{DH}	Data hold time	to be defined	30		
t_{ST}	Data shift timing	to be defined	0		
t_{STH}	Data shift hold time	to be defined	125		
t_{DFR}	Permissible frame signal delay	to be defined	-500	500	
t_{pD}	Data output delay time	to be defined	30	170	

LCD Timing

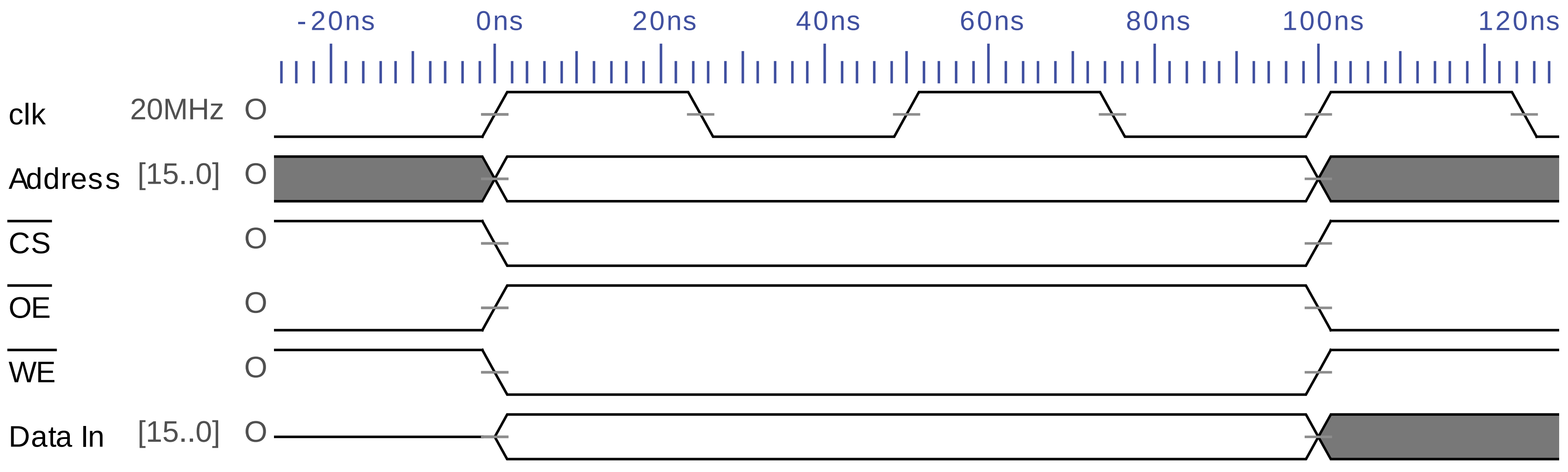


C:\Documents and Settings\Administrator\Desktop\LCD_Timing_Long.tdml

SRAM

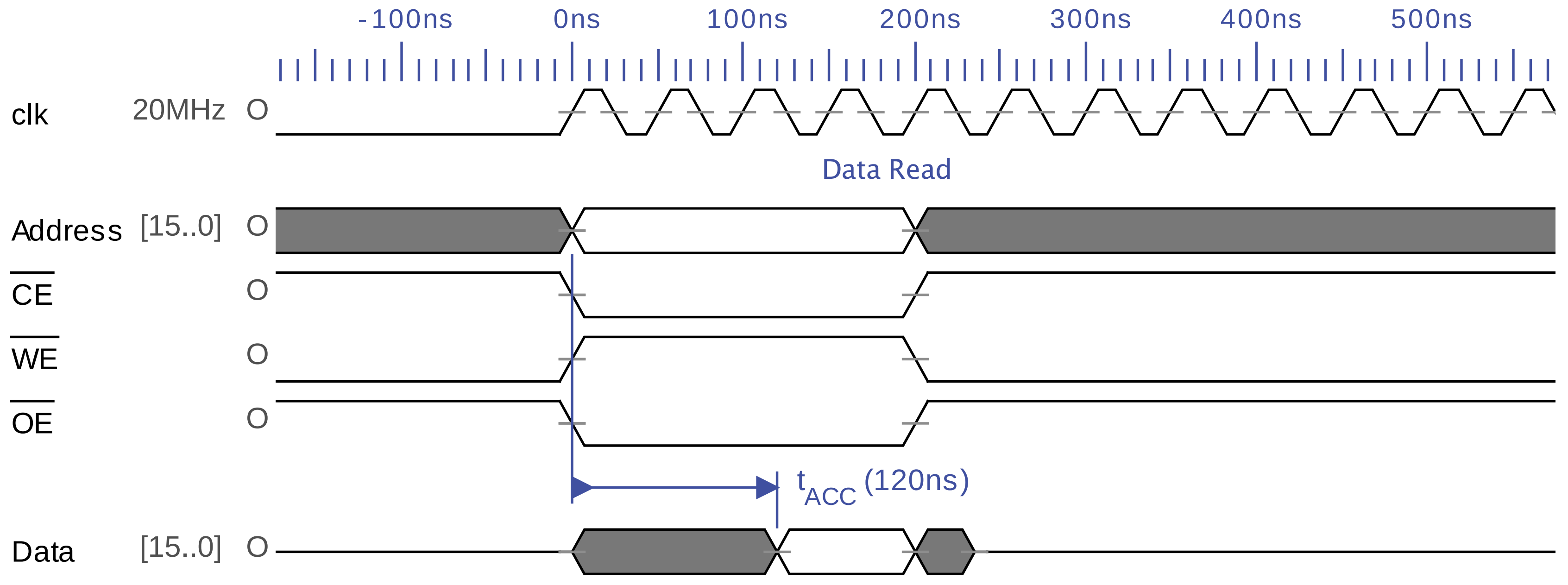

-50ns
0ps
50ns
100ns
150ns
200ns
250ns
300ns
clk
20MHz
O
Data Read
Address
[15..0]
O
CS
O
RD
O
T_AA
T_OH
Data
[15..0]
O
data_requirement

SRAM Write



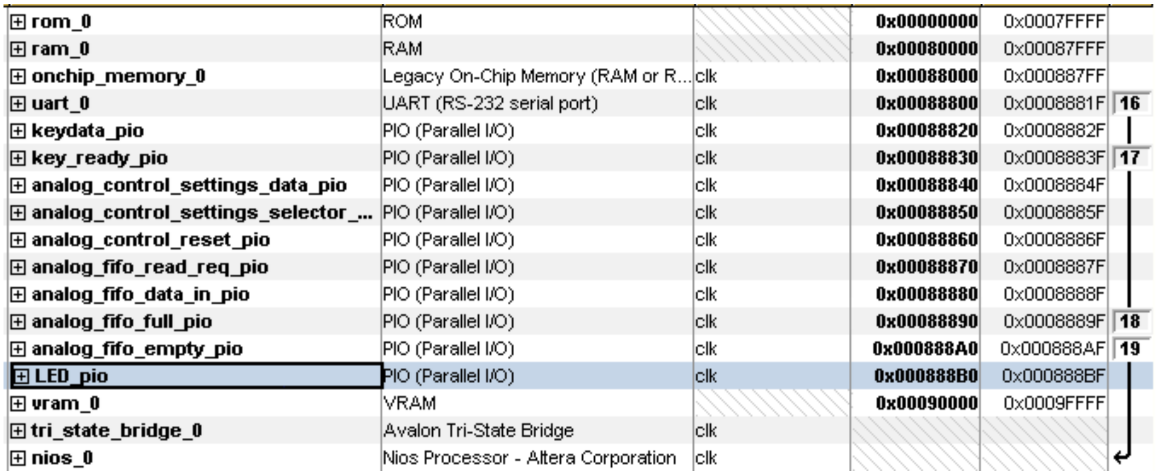
C:\Documents and Settings\Administrator\Desktop\from_windows\Timing Diagrams\SRAM Write.tdml

ROM Read



C:\Documents and Settings\Administrator\Desktop\from_windows\Timing Diagrams\ROM Read.tdml

Memory Map



⊞ **rom_0**	ROM		**0x00000000**	0x0007FFFF	
⊞ **ram_0**	RAM		**0x00080000**	0x00087FFF	
⊞ **onchip_memory_0**	Legacy On-Chip Memory (RAM or R...	clk	**0x00088000**	0x000887FF	
⊞ **uart_0**	UART (RS-232 serial port)	clk	**0x00088800**	0x0008881F	16
⊞ **keydata_pio**	PIO (Parallel I/O)	clk	**0x00088820**	0x0008882F	
⊞ **key_ready_pio**	PIO (Parallel I/O)	clk	**0x00088830**	0x0008883F	17
⊞ **analog_control_settings_data_pio**	PIO (Parallel I/O)	clk	**0x00088840**	0x0008884F	
⊞ **analog_control_settings_selector_...**	PIO (Parallel I/O)	clk	**0x00088850**	0x0008885F	
⊞ **analog_control_reset_pio**	PIO (Parallel I/O)	clk	**0x00088860**	0x0008886F	
⊞ **analog_fifo_read_req_pio**	PIO (Parallel I/O)	clk	**0x00088870**	0x0008887F	
⊞ **analog_fifo_data_in_pio**	PIO (Parallel I/O)	clk	**0x00088880**	0x0008888F	
⊞ **analog_fifo_full_pio**	PIO (Parallel I/O)	clk	**0x00088890**	0x0008889F	18
⊞ **analog_fifo_empty_pio**	PIO (Parallel I/O)	clk	**0x000888A0**	0x000888AF	19
⊞ **LED_pio**	PIO (Parallel I/O)	clk	**0x000888B0**	0x000888BF	
⊞ **vram_0**	VRAM		**0x00090000**	0x0009FFFF	
⊞ **tri_state_bridge_0**	Avalon Tri-State Bridge	clk			
⊞ **nios_0**	Nios Processor - Altera Corporation	clk			

Digital Oscilloscope APEX20K Pin Assignments

To	Location	General Function	Special Function
CLOCK	PIN_27	Dedicated Clock	CLK1
RESET	PIN_142	Row I/O	nRS
U4OE1	PIN_187	Column I/O	
U4OE2	PIN_206	Column I/O	
U6OE1	PIN_57	Column I/O	
U6OE2	PIN_84	Column I/O	
U6DIR2	PIN_85	Column I/O	
U7DIR1	PIN_87	Column I/O	
U7DIR2	PIN_108	Row I/O	
U7OE1	PIN_88	Column I/O	
U7OE2	PIN_107	Row I/O	
U8DIR1	PIN_2	Row I/O	
U8DIR2	PIN_31	Row I/O	
U8OE1	PIN_5	Row I/O	
U8OE2	PIN_30	Row I/O	
U9DIR1	PIN_32	Row I/O	
U9DIR2	PIN_55	Column I/O	
U9OE1	PIN_33	Row I/O	
U9OE2	PIN_54	Column I/O	
U10OE1	PIN_112	Row I/O	
U10OE2	PIN_158	Column I/O	
addr[0]	PIN_71	Column I/O	
addr[1]	PIN_72	Column I/O	
addr[2]	PIN_73	Column I/O	
addr[3]	PIN_74	Column I/O	
addr[4]	PIN_89	Column I/O	
addr[5]	PIN_90	Column I/O	
addr[6]	PIN_91	Column I/O	
addr[7]	PIN_92	Column I/O	
addr[8]	PIN_93	Column I/O	
addr[9]	PIN_94	Column I/O	
addr[10]	PIN_96	Column I/O	
addr[11]	PIN_97	Column I/O	
addr[12]	PIN_98	Column I/O	
addr[13]	PIN_99	Column I/O	
addr[14]	PIN_100	Column I/O	
addr[15]	PIN_101	Column I/O	
addr[16]	PIN_102	Column I/O	
addr[17]	PIN_103	Column I/O	
addr[18]	PIN_104	Column I/O	
data[0]	PIN_58	Column I/O	
data[1]	PIN_59	Column I/O	
data[2]	PIN_60	Column I/O	
data[3]	PIN_61	Column I/O	
data[4]	PIN_62	Column I/O	
data[5]	PIN_63	Column I/O	
data[6]	PIN_65	Column I/O	

data[7]	PIN_66	Column I/O	
nCS_RAM	PIN_37	Row I/O	
nCS_ROM	PIN_34	Row I/O	
nRD	PIN_67	Column I/O	
nWR	PIN_68	Column I/O	
U6DIR1	PIN_56	Column I/O	
TX1	PIN_17	Row I/O	
RX2	PIN_7	Row I/O	
keypad_rov	PIN_120	Row I/O	
keypad_rov	PIN_122	Row I/O	
keypad_rov	PIN_123	Row I/O	
keypad_rov	PIN_134	Row I/O	
keypad_col	PIN_140	Row I/O	
keypad_col	PIN_144	Row I/O	
keypad_col	PIN_151	Row I/O	
keypad_col	PIN_152	Row I/O	
U10DIR1	PIN_111	Row I/O	
U10DIR2	PIN_161	Column I/O	
STATUS_LE	PIN_117	Row I/O	
U4DIR1	PIN_179	Column I/O	
U4DIR2	PIN_207	Column I/O	
vaddr[0]	PIN_44	Row I/O	
vaddr[1]	PIN_204	Column I/O	
vaddr[2]	PIN_203	Column I/O	
vaddr[3]	PIN_202	Column I/O	
vaddr[4]	PIN_201	Column I/O	
vaddr[5]	PIN_200	Column I/O	
vaddr[6]	PIN_198	Column I/O	
vaddr[7]	PIN_197	Column I/O	
vdata[0]	PIN_193	Column I/O	
vdata[1]	PIN_194	Column I/O	
vdata[2]	PIN_195	Column I/O	
vdata[3]	PIN_196	Column I/O	
nRAS	PIN_45	Row I/O	
nCAS	PIN_46	Row I/O	
nDTOE	PIN_49	Row I/O	
nWBWE	PIN_50	Row I/O	
VRAM_SC	PIN_51	Row I/O	
DI	PIN_38	Row I/O	
DisplayEnal	PIN_40	Row I/O	
FR	PIN_41	Row I/O	
LP	PIN_113	Row I/O	
ECL	PIN_115	Row I/O	
YSCL	PIN_42	Row I/O	

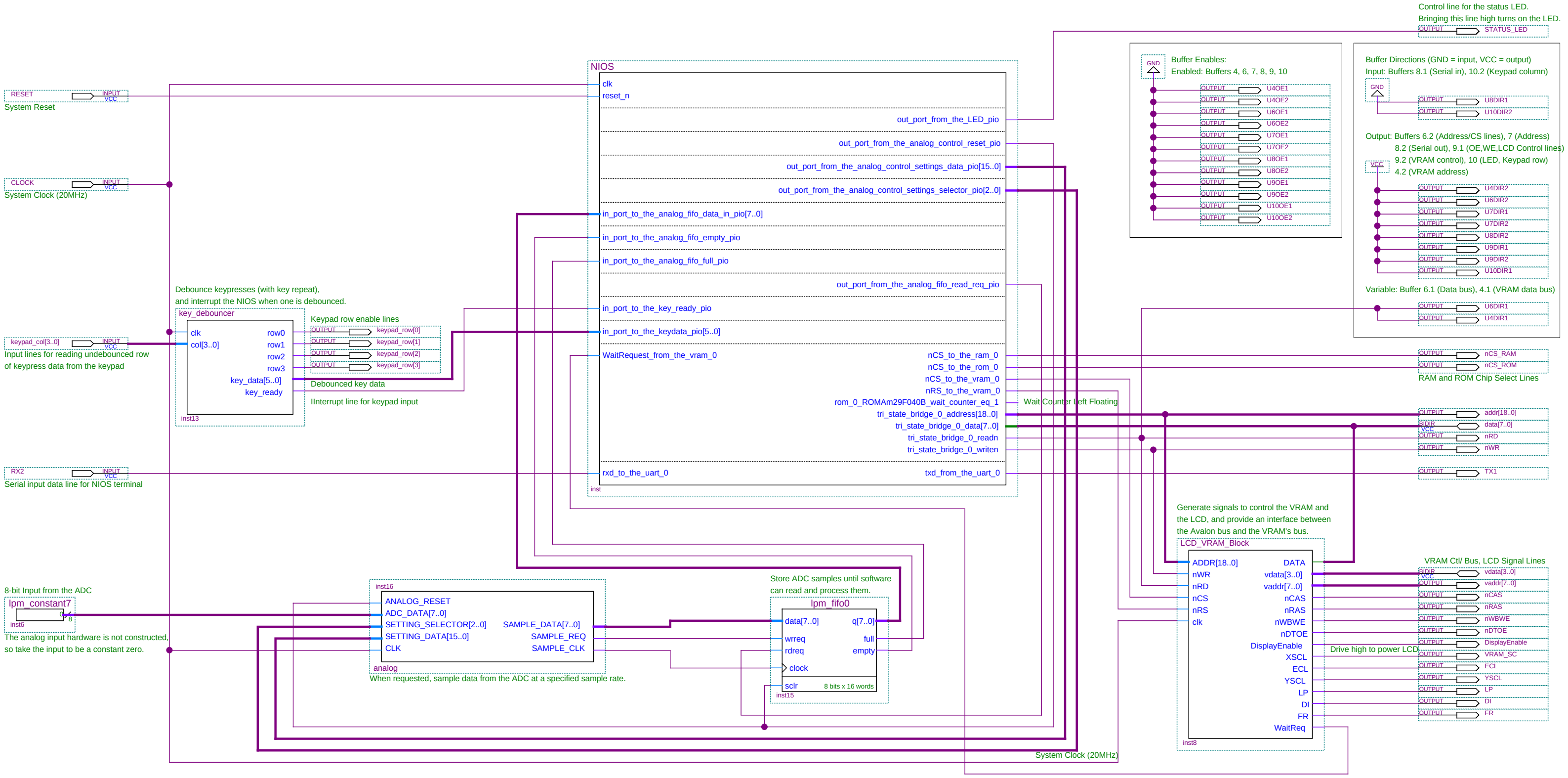
Quartus Design

- Top Level Design
- Analog Control Block
- LCD/VRAM Control Block
- LCD Signal Generator
- Keypad Debouncer
- Clock Divider

Date: November , 2008

oscilloscope.bdf

Project: oscilloscope



Control line for the status LED.
Bringing this line high turns on the LED.
STATUS_LED
RESET
System Reset
CLOCK
System Clock (20MHz)
keypad_col[3..0]
Input lines for reading undebounced row
of keypress data from the keypad
RX2
Serial input data line for NIOS terminal
INPUT VCC
OUTPUT
NIOS
clk
reset_n
out_port_from_the_LED_pio
out_port_from_the_analog_control_reset_pio
out_port_from_the_analog_control_settings_data_pio[15..0]
out_port_from_the_analog_control_settings_selector_pio[2..0]
in_port_to_the_analog_fifo_data_in_pio[7..0]
in_port_to_the_analog_fifo_empty_pio
in_port_to_the_analog_fifo_full_pio
out_port_from_the_analog_fifo_read_req_pio
in_port_to_the_key_ready_pio
in_port_to_the_keydata_pio[5..0]
WaitRequest_from_the_vram_0
nCS_to_the_ram_0
nCS_to_the_rom_0
nCS_to_the_vram_0
nRS_to_the_vram_0
rom_0_ROMAm29F040B_wait_counter_eq_1
tri_state_bridge_0_address[18..0]
tri_state_bridge_0_data[7..0]
tri_state_bridge_0_readn
tri_state_bridge_0_writen
rxd_to_the_uart_0
txd_from_the_uart_0
inst
Wait Counter Left Floating
Debounce keypresses (with key repeat),
and interrupt the NIOS when one is debounced.
key_debouncer
clk
col[3..0]
row0
row1
row2
row3
key_data[5..0]
key_ready
inst13
Keypad row enable lines
keypad_row[0]
keypad_row[1]
keypad_row[2]
keypad_row[3]
Debounced key data
!Interrupt line for keypad input
Buffer Enables:
Enabled: Buffers 4, 6, 7, 8, 9, 10
GND
U4OE1
U4OE2
U6OE1
U6OE2
U7OE1
U7OE2
U8OE1
U8OE2
U9OE1
U9OE2
U10OE1
U10OE2
Buffer Directions (GND = input, VCC = output)
Input: Buffers 8.1 (Serial in), 10.2 (Keypad column)
U8DIR1
U10DIR2
Output: Buffers 6.2 (Address/CS lines), 7 (Address)
8.2 (Serial out), 9.1 (OE,WE,LCD Control lines)
9.2 (VRAM control), 10 (LED, Keypad row)
4.2 (VRAM address)
VCC
U4DIR2
U6DIR2
U7DIR1
U7DIR2
U8DIR2
U9DIR1
U9DIR2
U10DIR1
Variable: Buffer 6.1 (Data bus), 4.1 (VRAM data bus)
U6DIR1
U4DIR1
nCS_RAM
nCS_ROM
RAM and ROM Chip Select Lines
addr[18..0]
BIDIR
data[7..0]
nRD
nWR
TX1
8-bit Input from the ADC
lpm_constant7
8
inst6
The analog input hardware is not constructed,
so take the input to be a constant zero.
inst16
ANALOG_RESET
ADC_DATA[7..0]
SETTING_SELECTOR[2..0]
SETTING_DATA[15..0]
CLK
SAMPLE_DATA[7..0]
SAMPLE_REQ
SAMPLE_CLK
analog
When requested, sample data from the ADC at a specified sample rate.
Store ADC samples until software
can read and process them.
lpm_fifo0
data[7..0]
wrreq
rdreq
clock
sclr
q[7..0]
full
empty
8 bits x 16 words
inst15
System Clock (20MHz)
Generate signals to control the VRAM and
the LCD, and provide an interface between
the Avalon bus and the VRAM's bus.
LCD_VRAM_Block
ADDR[18..0]
nWR
nRD
nCS
nRS
clk
DATA
vdata[3..0]
vaddr[7..0]
nCAS
nRAS
nWBWE
nDTOE
DisplayEnable
XSCL
ECL
YSCL
LP
DI
FR
WaitReq
inst8
Drive high to power LCD
VRAM Ctl/ Bus, LCD Signal Lines
vdata[3..0]
vaddr[7..0]
nCAS
nRAS
nWBWE
nDTOE
DisplayEnable
VRAM_SC
ECL
YSCL
LP
DI
FR

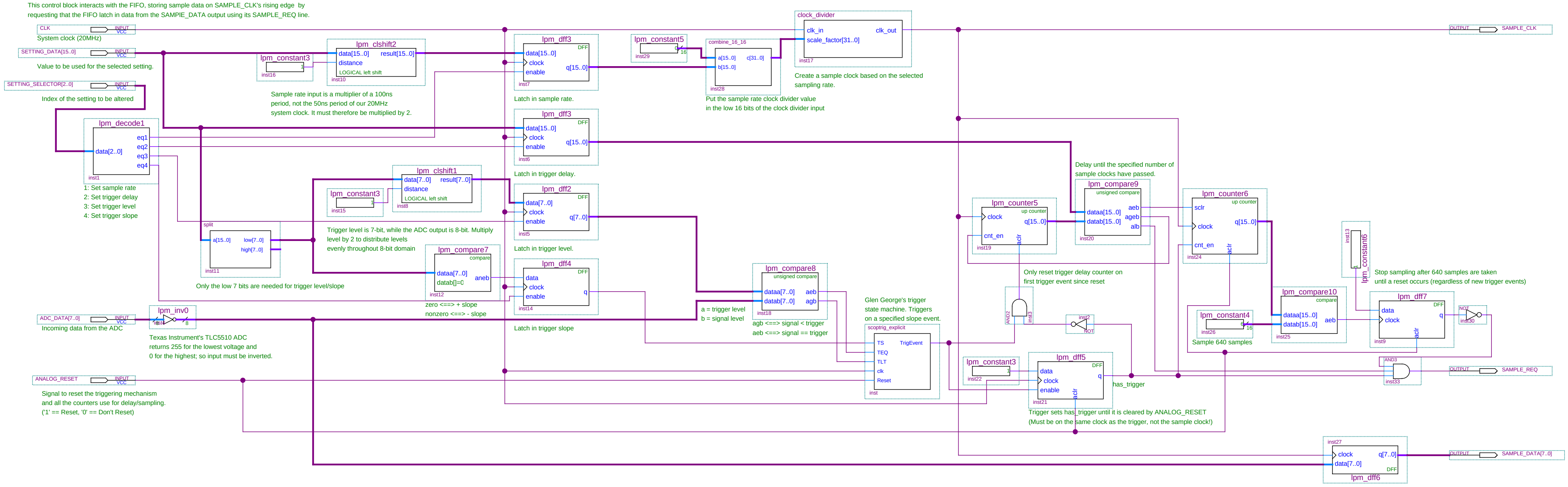
Revision:

Date: November , 2008

analog.bdf

Project: oscilloscope

Analog Control Block
Samples data from the ADC at a specified rate after a set delay
following a trigger event (defined by a configurable level and slope)
The settings are altered by first selecting the desired setting by placing
a 3-bit identifier on the selector line:
1 == Sample rate, 2 == Trigger delay, 3 == Trigger level, 4 == Trigger slope
Then the value for that setting is placed on the SETTING_DATA line.
Once the setting is made, the setting can be deselected by writing the identifier 0
or by selecting another setting to change.
This control block interacts with the FIFO, storing sample data on SAMPLE_CLK's rising edge by
requesting that the FIFO latch in data from the SAMPlE_DATA output using its SAMPLE_REQ line.



System clock (20MHz)

Value to be used for the selected setting.

Index of the setting to be altered

1: Set sample rate
2: Set trigger delay
3: Set trigger level
4: Set trigger slope

Sample rate input is a multiplier of a 100ns
period, not the 50ns period of our 20MHz
system clock. It must therefore be multiplied by 2.

Latch in sample rate.

Put the sample rate clock divider value
in the low 16 bits of the clock divider input

Create a sample clock based on the selected
sampling rate.

Latch in trigger delay.

Trigger level is 7-bit, while the ADC output is 8-bit. Multiply
level by 2 to distribute levels
evenly throughout 8-bit domain

Latch in trigger level.

Only the low 7 bits are needed for trigger level/slope

zero <==> + slope
nonzero <==> - slope

Latch in trigger slope

Incoming data from the ADC

Texas Instrument's TLC5510 ADC
returns 255 for the lowest voltage and
0 for the highest; so input must be inverted.

a = trigger level
b = signal level

agb <==> signal < trigger
aeb <==> signal == trigger

Glen George's trigger
state machine. Triggers
on a specified slope event.

Signal to reset the triggering mechanism
and all the counters use for delay/sampling.
('1' == Reset, '0' == Don't Reset)

Delay until the specified number of
sample clocks have passed.

Only reset trigger delay counter on
first trigger event since reset

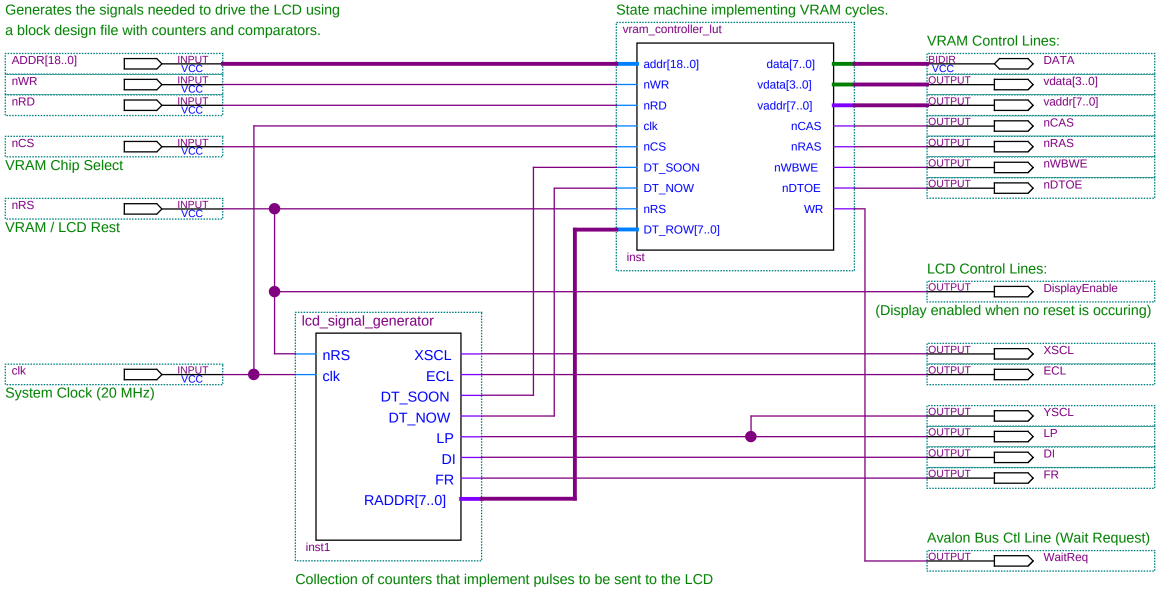
Stop sampling after 640 samples are taken
until a reset occurs (regardless of new trigger events)

Sample 640 samples

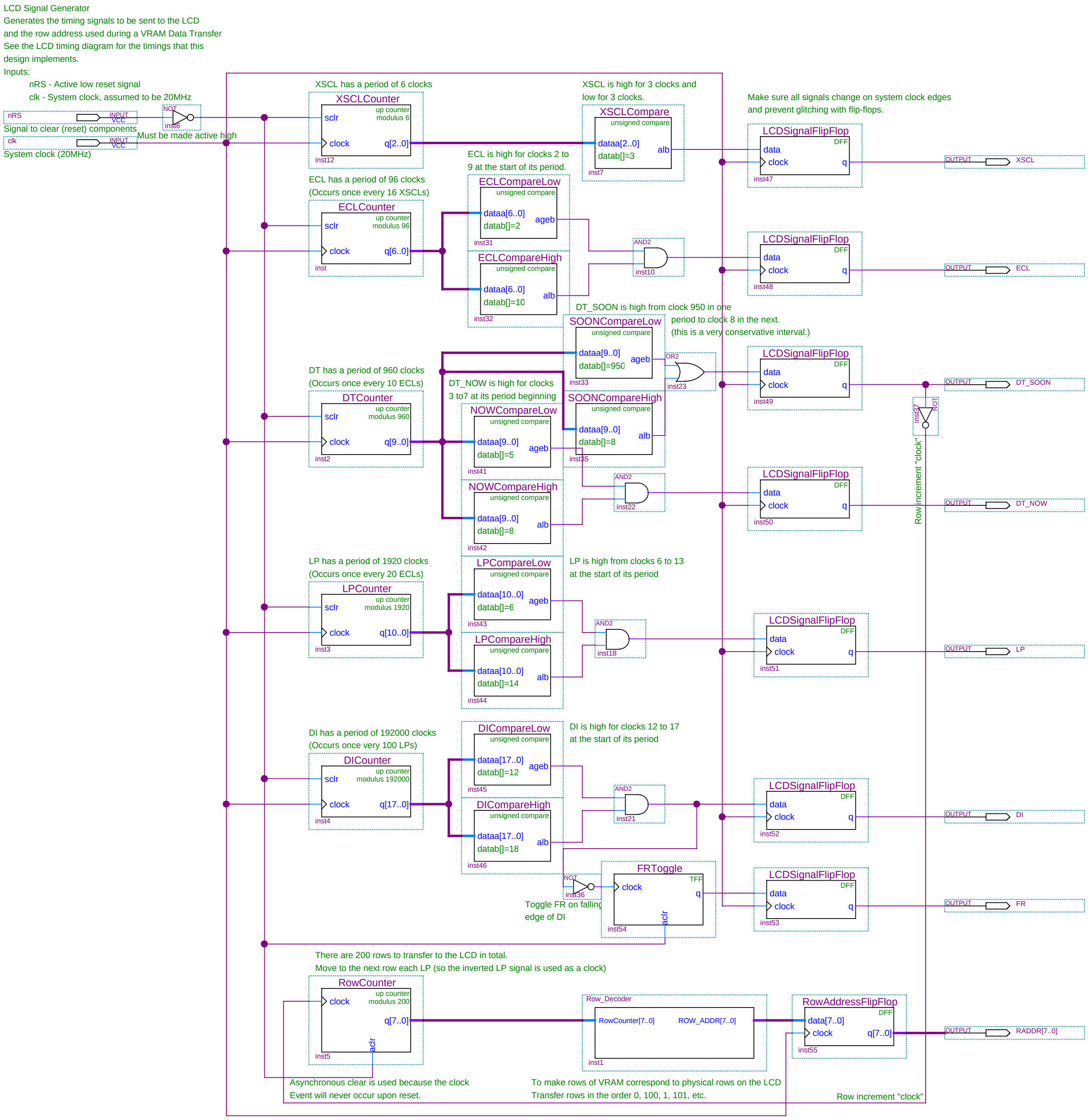
Trigger sets has_trigger until it is cleared by ANALOG_RESET
(Must be on the same clock as the trigger, not the sample clock!)

LCD/VRAM Block
Implements reads, writes, data transfers, and refreshes
for the VRAM using a finite state machine implemented in VHDL.
Generates the signals needed to drive the LCD using
a block design file with counters and comparators.

State machine implementing VRAM cycles.



Collection of counters that implement pulses to be sent to the LCD
and computes the address of the row the VRAM needs to transfer
into its serial shift register.



LCD Signal Generator
Generates the timing signals to be sent to the LCD
and the row address used during a VRAM Data Transfer
See the LCD timing diagram for the timings that this
design implements.
Inputs:
nRS - Active low reset signal
clk - System clock, assumed to be 20MHz

Signal to clear (reset) components

Must be made active high

System clock (20MHz)

XSCL has a period of 6 clocks

XSCL is high for 3 clocks and
low for 3 clocks.

Make sure all signals change on system clock edges
and prevent glitching with flip-flops.

ECL is high for clocks 2 to
9 at the start of its period.

ECL has a period of 96 clocks
(Occurs once every 16 XSCLs)

DT_SOON is high from clock 950 in one
period to clock 8 in the next.
(this is a very conservative interval.)

DT has a period of 960 clocks
(Occurs once every 10 ECLs)

DT_NOW is high for clocks
3 to7 at its period beginning

LP has a period of 1920 clocks
(Occurs once every 20 ECLs)

LP is high from clocks 6 to 13
at the start of its period

DI has a period of 192000 clocks
(Occurs once very 100 LPs)

DI is high for clocks 12 to 17
at the start of its period

Toggle FR on falling
edge of DI

There are 200 rows to transfer to the LCD in total.
Move to the next row each LP (so the inverted LP signal is used as a clock)

Asynchronous clear is used because the clock
Event will never occur upon reset.

To make rows of VRAM correspond to physical rows on the LCD
Transfer rows in the order 0, 100, 1, 101, etc.

Date: November , 2008

key_debouncer.bdf

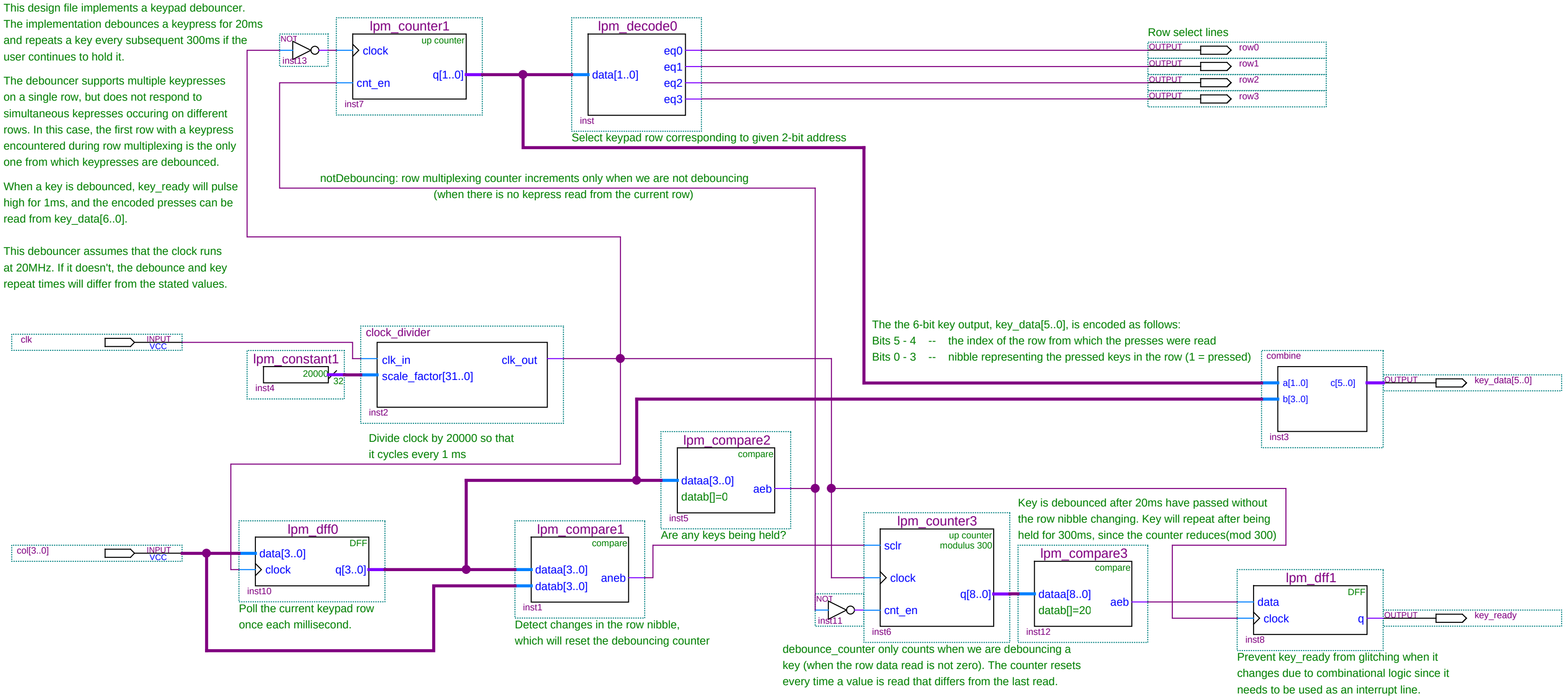
Project: oscilloscope

This design file implements a keypad debouncer. The implementation debounces a keypress for 20ms and repeats a key every subsequent 300ms if the user continues to hold it.

The debouncer supports multiple keypresses on a single row, but does not respond to simultaneous kepresses occuring on different rows. In this case, the first row with a keypress encountered during row multiplexing is the only one from which keypresses are debounced.

When a key is debounced, key_ready will pulse high for 1ms, and the encoded presses can be read from key_data[6..0].

This debouncer assumes that the clock runs at 20MHz. If it doesn't, the debounce and key repeat times will differ from the stated values.



Select keypad row corresponding to given 2-bit address

notDebouncing: row multiplexing counter increments only when we are not debouncing
(when there is no kepress read from the current row)

The the 6-bit key output, key_data[5..0], is encoded as follows:
Bits 5 - 4 -- the index of the row from which the presses were read
Bits 0 - 3 -- nibble representing the pressed keys in the row (1 = pressed)

Divide clock by 20000 so that
it cycles every 1 ms

Are any keys being held?

Key is debounced after 20ms have passed without the row nibble changing. Key will repeat after being held for 300ms, since the counter reduces(mod 300)

Poll the current keypad row once each millisecond.

Detect changes in the row nibble, which will reset the debouncing counter

debounce_counter only counts when we are debouncing a key (when the row data read is not zero). The counter resets every time a value is read that differs from the last read.

Prevent key_ready from glitching when it changes due to combinational logic since it needs to be used as an interrupt line.

Revision:

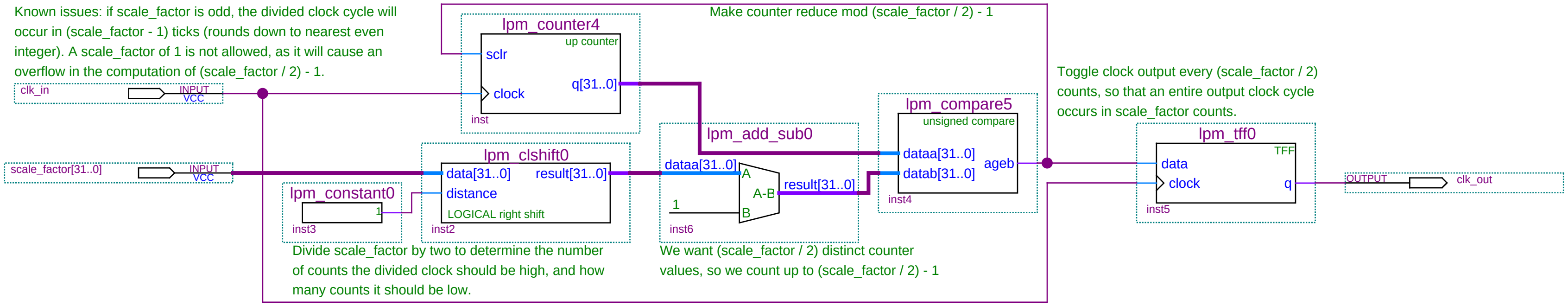
Date: November , 2008

clock_divider.bdf

Project: oscilloscope

Clock Divider

Divides clk_in by scale_factor, generating a new clock on clk_out.



Known issues: if scale_factor is odd, the divided clock cycle will occur in (scale_factor - 1) ticks (rounds down to nearest even integer). A scale_factor of 1 is not allowed, as it will cause an overflow in the computation of (scale_factor / 2) - 1.

Make counter reduce mod (scale_factor / 2) - 1

Toggle clock output every (scale_factor / 2) counts, so that an entire output clock cycle occurs in scale_factor counts.

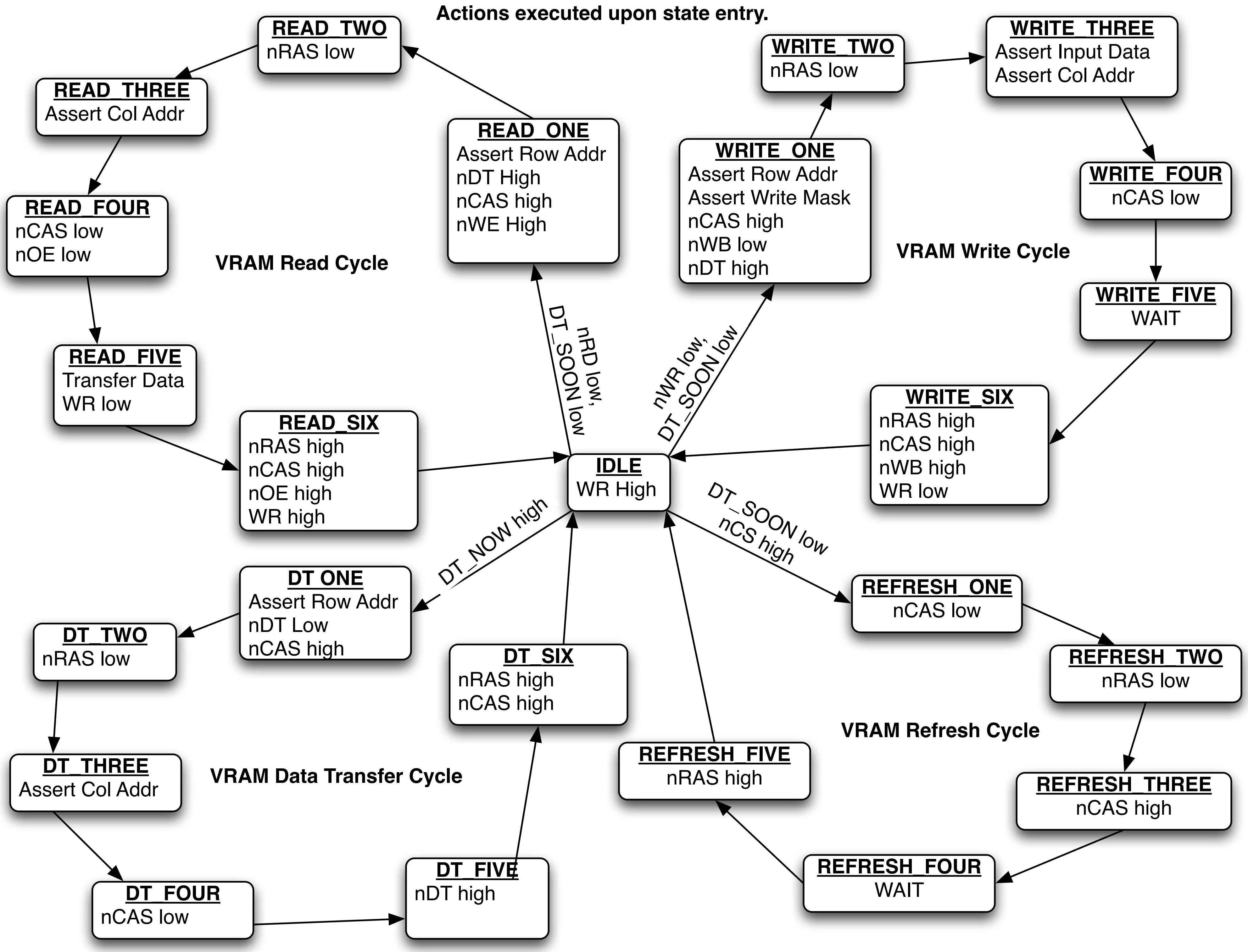
Divide scale_factor by two to determine the number of counts the divided clock should be high, and how many counts it should be low.

We want (scale_factor / 2) distinct counter values, so we count up to (scale_factor / 2) - 1

Revision:

VHDL

- VRAM Controller State Machine
  - State machine diagram
  - vram_controller_lut.vhd
- Miscellaneous Helper Blocks
  - split.vhd
  - combine.vhd
  - combine_16_16.vhd



Actions executed upon state entry.

```vhdl
--------------------------------------------------------------------------------
--
-- Oscilloscope VRAM Controller
--
-- This VHDL file implements a controller for the NEC uPD41264-12 VRAM chip.
-- The controller is implemeted using a state machine whose transitions are
-- a function of the input lines. Output signals change upon transition to
-- a new state, and their values are looked up in a table indexed by current
-- state number.
--
-- The controller uses a wait request line to halt the NIOS cpu until it is
-- able to complete a read or write, since a refresh cycle or data transfer
-- cycle could delay read/write cycles an amount unpredictable to the Avalon
-- Tristate Bus.
--
-- See the state diagram for a visual overview of the state machine.
-- Inputs and outputs are enumerated and described in the port section below.
--
-- Revision History:
--      10/23/08    Julian Panetta  Initial Revision
--      11/16/08    Julian Panetta  Reformatted to fit within 80 chracters wide
--
--------------------------------------------------------------------------------

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity vram_controller_lut is
    port    (
        -- 16-bit address input (row & column addresses) with 3 disregarded
        -- most significant bits so it interfaces with 19-bit shared Avalon bus
        addr    :   in std_logic_vector(18 downto 0);

        -- 8-bit data bus (data / writemask nibbles)
        data    :   inout std_logic_vector(7 downto 0);

        -- Write enable
        nWR     :   in std_logic;

       -- Output enable
        nRD     :   in std_logic;

        -- System clock (Assumed to be 20MHz)
        clk     :   in std_logic;

        -- Chip select
        nCS     :   in std_logic;

        -- Data transfer needs to occur before a new cycle can be completed.
        -- ('1' == true, '0' == false)
        DT_SOON :   in std_logic;

        -- Data transfer needs to occur NOW
        -- ('1' == true, '0' == false)
        DT_NOW  :   in std_logic;

        -- reset state machine to known, IDLE state
        nRS     :   in std_logic;

        -- Row address to be used for the data transfer cycle.
        -- (The data transfer column is always 0.)
        DT_ROW  :   in std_logic_vector(7 downto 0);
        -- 4-bit data bus to VRAM
        vdata   :   inout std_logic_vector(3 downto 0);

        -- 8-bit address bus out to VRAM
        vaddr   :   out std_logic_vector(7 downto 0);

        -- CAS signal to VRAM
        nCAS    :   out std_logic;

        -- RAS signal to VRAM
        nRAS    :   out std_logic;

        -- nWB/nWE signal to VRAM
        nWBWE   :   out std_logic;

        -- nDT/nOE signal to VRAM
        nDTOE   :   out std_logic;

        -- Wait request signal for NIOS
        WR      :   out std_logic
    );
end vram_controller_lut;

architecture Moore_machine of vram_controller_lut is
    constant    IDLE            :   integer range 0 to 23 := 0;
    -- VRAM Read Cycle States
    constant    READ_ONE        :   integer range 0 to 23 := 1;
    constant    READ_TWO        :   integer range 0 to 23 := 2;
    constant    READ_THREE      :   integer range 0 to 23 := 3;
    constant    READ_FOUR       :   integer range 0 to 23 := 4;
    constant    READ_FIVE       :   integer range 0 to 23 := 5;
    constant    READ_SIX        :   integer range 0 to 23 := 6;
    -- VRAM Write Cycle States
    constant    WRITE_ONE       :   integer range 0 to 23 := 7;
    constant    WRITE_TWO       :   integer range 0 to 23 := 8;
    constant    WRITE_THREE     :   integer range 0 to 23 := 9;
    constant    WRITE_FOUR      :   integer range 0 to 23 := 10;
    constant    WRITE_FIVE      :   integer range 0 to 23 := 11;
    constant    WRITE_SIX       :   integer range 0 to 23 := 12;
    -- VRAM Refresh Cycle States
    constant    REFRESH_ONE     :   integer range 0 to 23 := 13;
    constant    REFRESH_TWO     :   integer range 0 to 23 := 14;
    constant    REFRESH_THREE   :   integer range 0 to 23 := 15;
    constant    REFRESH_FOUR    :   integer range 0 to 23 := 16;
    constant    REFRESH_FIVE    :   integer range 0 to 23 := 17;
    -- VRAM Data Transfer Cycle States
    constant    DTRAN_ONE       :   integer range 0 to 23 := 18;
    constant    DTRAN_TWO       :   integer range 0 to 23 := 19;
    constant    DTRAN_THREE     :   integer range 0 to 23 := 20;
    constant    DTRAN_FOUR      :   integer range 0 to 23 := 21;
    constant    DTRAN_FIVE      :   integer range 0 to 23 := 22;
    constant    DTRAN_SIX       :   integer range 0 to 23 := 23;

    -- LUT for control line ouputs. 0 => nRAS, 1 => nCAS, 2 => nWBWE, 3 => nDTOE, 4 =>
 WR
    type TABLE is array(0 to 23) of std_logic_vector(4 downto 0);
    constant    OUTPUT_BITS     :   TABLE := (
                -- IDLE control line values
                "11111",

                -- READ_* control line values
                "11111", "11110", "11110", "10100", "00100", "11111",
                -- WRITE_* control line values
                "11011", "11010", "11010", "11000", "11000", "01111",

                -- REFRESH_* control line values
                "11101", "11100", "11100", "11100", "11111",

                -- DTRAN_* control line values
                "10111", "10110", "10110", "10100", "11100", "11111");

    signal      CurrentState    :   integer range 0 to 23;   -- current state
    signal      NextState       :   integer range 0 to 23;   -- next state
begin
    -- Compute the next state (function of current state and inputs) using
    -- concurrent statements
    NextState <=
        -- Transitions to IDLE
                IDLE        when    (nRS = '0')
        else    IDLE        when    (CurrentState = READ_SIX or
                     CurrentState = WRITE_SIX or CurrentState = REFRESH_FIVE or
                     CurrentState = DTRAN_SIX)

        -- Idle State transitions
        else    READ_ONE    when    (CurrentState = IDLE and nRD = '0' and
                                     DT_SOON = '0' and nCS = '0')
        else    WRITE_ONE   when    (CurrentState = IDLE and nWR = '0' and
                                     DT_SOON = '0' and nCS = '0')
        else    REFRESH_ONE when    (CurrentState = IDLE and nCS = '1' and
                                     DT_SOON = '0')
        else    DTRAN_ONE   when    (CurrentState = IDLE and DT_NOW = '1')
        else    IDLE        when    (CurrentState = IDLE)
        -- Movement Within Cycles
        else    CurrentState + 1;

    -- Upon transition into the new state, update the contol line values
    -- using the output lookup table.
    output_computation : process (CurrentState)
    begin
        -- Look up and set control line values for the current state
        nRAS    <= OUTPUT_BITS(CurrentState)(0);
        nCAS    <= OUTPUT_BITS(CurrentState)(1);
        nWBWE   <= OUTPUT_BITS(CurrentState)(2);
        nDTOE   <= OUTPUT_BITS(CurrentState)(3);
        WR      <= OUTPUT_BITS(CurrentState)(4);

        -- VRAM address line access
        -- Assert row address for the first and second read and write states.
        if (CurrentState = READ_ONE or CurrentState = WRITE_ONE or
            CurrentState = READ_TWO or CurrentState = WRITE_TWO) then
            vaddr   <= addr(15 downto 8);
        else
            -- Assert col address for the third/fourth read and write states.
            if (CurrentState = READ_THREE or CurrentState = READ_FOUR or
                CurrentState = WRITE_THREE or CurrentState = WRITE_FOUR) then
                vaddr   <= addr(7 downto 0);
            else
                -- Assert DT row address for the first and second DT states.
                if (CurrentState = DTRAN_ONE or CurrentState = DTRAN_TWO) then
                    vaddr   <= DT_ROW;
                else
                    -- Output zero addess in all other states.
                    vaddr   <= "00000000";
                end if;
            end if;
        end if;
        -- VRAM data bus access
        -- Transfer VRAM data to Avalon Bus for the fifth read state.
        if (CurrentState = READ_FIVE) then
            data    <= "0000" & vdata;
        else
            -- Assert write mask for the first and second write state.
            if (CurrentState = WRITE_ONE or CurrentState = WRITE_TWO) then
                vdata   <= data(7 downto 4);
            else
                -- Assert graphics nibble for the third and fourth write state.
                if (CurrentState = WRITE_THREE or
                    CurrentState = WRITE_FOUR) then
                    vdata   <= data(3 downto 0);
                else
                    -- In all other states, the Avalon data bus and the VRAM
                    -- data bus should be high impedance.
                    vdata <= "ZZZZ";
                    data <= "ZZZZZZZZ";
                end if;
            end if;
        end if;

    end process output_computation;

    -- Make actual transition to the new state on rising clock edge
    make_transition : process (clk)
    begin
            if clk = '1' then            -- Only change on the rising clock edge
                CurrentState <= NextState;      -- Transition to the new state
            end if;
    end process;

end Moore_machine;
```

```
------------------------------------------------------------------------------
--
-- Split
--
-- Splits the 16-bit input bits read from port a evenly into the 8 LSBs (place
d
-- on "low" ouptut line) and the 8 MSBs (placed on "high" output line).
--
-- Revision History:
--      09/05/08    Julian Panetta  Initial Revision
------------------------------------------------------------------------------
Library ieee;
use ieee.std_logic_1164.all;

entity split is
    port    (
        a       :   in  std_logic_vector(15 downto 0);
        low     :   out std_logic_vector(7 downto 0);
        high    :   out std_logic_vector(7 downto 0)
);
end split;

architecture    dataflow    of  split   is
begin
    low <= a(7 downto 0);
    high <= a(15 downto 8);
end dataflow;
```

```
------------------------------------------------------------------------------
--
-- Combine
--
-- (For use with computing debounced keypress output)
-- Concatenates 2-bit row address read from port a with 4-bit column data read
-- from port b into a 6-bit output placed on port c. The row address is placed
-- in the MSBs of the output and the column address is placed in the LSBs.
--
------------------------------------------------------------------------------
Library ieee;
use ieee.std_logic_1164.all;

entity combine is
    port    (
        a       :   in  std_logic_vector(1 downto 0);
        b       :   in  std_logic_vector(3 downto 0);
        c       :   out std_logic_vector(5 downto 0)
);
end combine;

architecture    dataflow    of  combine is
begin
    c <= a & b;
end dataflow;
```

```
-----------------------------------------------------------------------------
--
-- Combine 16_16
--
-- Concatenates two 16-bit inputs (read from ports a and b) into a 32-bit
-- output value placed on port c. the value of port a is placed in the most
-- significant bits of c, and b's value is placed in the least significant
-- bits.
--
-- Revision History:
--      09/05/08    Julian Panetta  Initial Revision
-----------------------------------------------------------------------------
Library ieee;
use ieee.std_logic_1164.all;

entity combine_16_16 is
    port    (
        a       :   in  std_logic_vector(15 downto 0);
        b       :   in  std_logic_vector(15 downto 0);
        c       :   out std_logic_vector(31 downto 0)
);
end combine_16_16;

architecture    dataflow    of  combine_16_16   is
begin
    c <= a & b;
end dataflow;
```

Software

- Drivers/Initialization (NIOS Assembly)
  - start.s
    - Initializes hardware and calls oscilloscope operating system main loop.
    - Global symbols: _start
  - keypad.s
    - Interrupt-driven driver for the keypad
    - Global symbols: install_key_handler, getkey, key_available
  - display.s
    - Code for drawing on and clearing the LCD
    - Global symbols: plot_pixel, clear_display
  - interrupts.s
    - Code for initializing interrupts
    - Global symbols: init_interrupts
  - constants.s
    - Include file holding constants used throughout the NIOS assembly files listed above.
- OS (C)
  - stubfuncs.c
    - Functions emulating hardware not yet implemented
    - Global symbols: set_sample_rate, set_trigger, set_delay, start_sample, sample_done
  - interfac.h
    - Include file holding constants used throughout the oscilloscope operating system.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; File: start.s
; Description:
;   Code run when the NIOS resets. Performs hardware initialization and then
;   branches to Glen George's oscilloscope code.
;
; Input:    Keypresses from the user, analog signal input (not implemented)
; Output:   Output displayed
;
; Known Bugs:   None
; Revision History:
;   11/04/08    Julian Panetta  Initial Revision
;   11/17/08    Julian Panetta  Added more extensive documentation
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.include "excalibur.s"
.include "constants.s"

.text
.global _start

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; void _start(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Code located at the NIOS' reset location. Initializes hardware and then
;   runs the oscilloscope OS.
; Operation:
;   - Installs the keypad handler, initialize_interrupts, and forces interrupts
;     to be enabled.
;   - Turns on the status LED.
;   - Branches to the oscilloscope's main loop from which the NIOs should never
;     return.
;
; Arguments: None
; Return Values: None
;
; Inputs:   Keypresses read from keypad
; Outputs:  User interface displayed on LCD
;
; Registers Destroyed: None (Registers SAVED, no supervisor/caller anyway).
; Shared Variables: None
;
; Data Structures/Algorithms: None
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
_start:
    SAVE    %sp, 0      ; No stack space is needed

    _BSR    install_key_handler
    NOP

    _BSR    init_interrupts
    NOP

    ; Turn on interrupts
    ; According to the NIOS Programmer's Manual, interrupts should be enabled
    ; by default and need only be enabled if explicitly disabled previously.
    ; However, I have found the interrupts don't function if they aren't
    ; enabled as done by the following two lines. These lines were copied
    ; verbatim from the Programmer's Manual.
    PFX     9
    WRCTL   %g0

    ; Turn on the LED to indicate the system has booted
    MOVI    %l1, LED_ON
    MOVIA   %l0, na_LED_pio
    PFX     np_piodata
    ST      [%l0], %l1

    _BSR    main    ; Run Glen's oscilloscope code
    NOP

    ; This line should never be reached.
    RESTRET
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; File: keypad.s
; Description:
;       Routines to allow Glen George's oscilloscope software interface with
;       the debounced keypad.
; Input:    Debounced keypresses from the user (ISR run when keypress made).
; Output:   None.
;
; Known Bugs:   None
; Revision History:
;       09/06/08    Julian Panetta  Initial Revision
;       11/17/08    Julian Panetta  Added documentation and replaced
;                                   magic numbers with constants.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.include "excalibur.s"
.include "constants.s"

.data
key_ready:  .4byte  false
key_data:   .4byte  KEY_ILLEGAL

.text
.global install_key_handler
.global getkey
.global key_available

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; void install_key_handler(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Installs and initializes the isr that is run when a key is debounced.
; Operation:
;   - Intalls ISR.
;   - Clears key_ready and key_data variables.
;   - Clears edge capture register so previous keypresses aren't detected.

; Arguments: None
; Return Values: None
;
; Outputs: None
; Registers Destroyed: None (Registers SAVED/RESTORED)
;
; Shared Variables:
;   key_ready   -- (Cleared) Boolean value recording the occurance of a keypress
;   key_data    -- (Cleared) Byte recording the value of the keypress
;
; Data Structures/Algorithms: None
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
install_key_handler:
    SAVE    %sp, 0  ; no stack space is needed

    MOVIA   %o0, na_key_ready_pio_irq   ; select key_ready interrupt
    MOVIA   %o1, key_handler@h          ; load ISR address
    MOVIA   %o2, 0                      ; context (ignored)

    BSR     nr_installuserisr           ; install our keypad isr
    NOP

    ; Initialize key_ready and key_data words to zero
    MOVIA   %l2, false

    MOVIA   %l1, key_ready
    ST      [%l1], %l2                  ; No key is ready initially
    MOVIA   %l2, KEY_ILLEGAL            ; (Not needed as false == KEY_ILLEGAL)
    MOVIA   %l1, key_data               ; No valid key is encoded initially
    ST      [%l1], %l2

    ; Clear the edge capture register to ensure a keypress made
    ; before loading the code doesn't result in an interrupt.
    MOVIA   %l1, na_key_ready_pio
    MOVI    %l0, false
    PFX     np_pioedgecapture
    ST      [%l1], %l0

    RESTRET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; void key_handler(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   ISR that is run when a key is pressed. Simply records the presence of a key
;   and resets the interrupt event.
; Operation:
;   - Clears the edge capture register so the event doesn't fire until a new
;     keypress is made.
;   - Sets key_ready and stores debounced key data in key_data.
;
; Arguments: None
; Return Values: None
;
; Inputs:   Keys pressed by the user trigger this handler
; Outputs:  None
;
; Registers Destroyed: None (Registers SAVED/RESTORED)
; Shared Variables:
;   key_ready   -- (Set) Boolean value recording the occurance of a keypress
;   key_data    -- (Stored) Byte recording the value of the keypress
;
; Data Structures/Algorithms: None
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
key_handler:
    SAVE    %sp, 0          ; no stack space is needed
    ; clear the edge capture register
    MOVIA   %l1, na_key_ready_pio
    MOVI    %l0, 0
    PFX     np_pioedgecapture
    ST      [%l1], %l0

    MOVIA   %l0, key_ready
    MOVI    %r0, true
    ST8S    [%l0], %r0, 0   ; Indicate a debounced key was received

    MOVIA   %l0, na_keydata_pio
    PFX     np_piodata
    LD      %l1, [%l0]

    MOVIA   %l0, key_data
    ST      [%l0], %l1

    RESTORE
    TRET    %o7

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; unsigned char key_available(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Returns 0x01 if a key press is ready for reading and 0x00 if not.
; Operation:
;   - Reads key_ready and extracts bit 1 as the return value.
;
; Arguments: None
; Return Values: None
;
; Inputs: None
; Outputs: None
;
; Registers Destroyed: None (Registers SAVED/RESTORED)
; Shared Variables:
;   key_ready   -- (Read) Boolean value recording the occurance of a keypress
;
; Data Structures/Algorithms: None
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
key_available:
    SAVE    %sp, 0          ; No stack space is needed
    MOVIA   %o1, key_ready
    LD      %o0, [%o1]
    MOVI    %o1, true

    AND     %o0, %o1        ; Check if a keypress is ready
    RESTRET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; int getkey(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Waits until a valid keypress is made (if one hasn't been already) and then
;   returns that keypress' code.
; Operation:
;   - Waits until a a keypress has been made.
;   - Performs a lookup to see if that key is valid.
;   - Returns the key if it is valid, otherwise waits for another keypress.
;
; Arguments: None
; Return Values: None
;
; Inputs: Valid keys pressed by the user are returned by this routine.
; Outputs: None
;
; Registers Destroyed: None (Registers SAVED/RESTORED)
; Shared Variables: None
;   key_ready   -- (Read) Boolean value recording the occurance of a keypress
;   key_data    -- (Read) Byte recording the value of the keypress
;
; Data Structures/Algorithms: Look up table is used to validate keypresses
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
getkey:
    SAVE    %sp, 0          ; no stack space is needed

    MOVIA   %l0, key_ready
    MOVIA   %l3, key_data
    MOVI    %l2, true
wait:
    LD      %l1, [%l0]
    AND     %l1, %l2        ; Check key is ready

    IFS     cc_z
    BR      wait            ; wait until a key is ready
    NOP

    MOVI    %l1, false
    ST      [%l0], %l1      ; clear key_ready
    LD      %i0, [%l3]      ; get key_data
    EXT8D   %i0, %l1        ; put key_data in low byte of return value

    MOVIA   %l4, valid_keys
validate_loop:
    LD      %l5, [%l4]

    CMP     %i0, %l5
    IFS     cc_eq           ; valid key?
    BR      valid           ; return it.
    NOP

    CMPI    %l5, KEY_ILLEGAL
    IFS     cc_eq           ; invalid key?
    BR      wait            ; wait for a valid one.
    NOP

    BR      validate_loop   ; not sure yet!
    ADDI    %l4, 1          ; move to the next key entry

valid:
    RESTRET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; valid_keys:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Constant table of valid keypress values terminated by KEY_ILLEGAL.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
valid_keys:
    .word   KEY_UP
    .word   KEY_DOWN
    .word   KEY_LEFT
    .word   KEY_RIGHT
    .word   KEY_MENU
    .word   KEY_ILLEGAL ; KEY_ILLEGAL Terminates the list of valid keys
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; File: display.s
; Description:
;       Routines to set individual pixels on the LCD and clear the entire LCD.
; Input:        None.
; Output:       Pixels displayed on LCD.
;
; Known Bugs:   None
; Revision History:
;       11/04/08    Julian Panetta  Initial Revision
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.include "excalibur.s"
.include "constants.s"

.text
.global plot_pixel
.global clear_display

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; void plot_pixel(unsigned int x, unsigned int y, int p)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Sets the pixel at the passed (x, y) to the value p.
;   The origin of the screen coordinate system is the upper left corner.
; Operation:
;   - Computes the VRAM address of the nibble holding the desired pixel.
;   - Adds this offset to the VRAM base address to compute the absolute address.
;   - Creates a bit mask for the VRAM write so that only the desired pixel of
;     the nibble is modified.
;   - Sets the write data to all 1's for p == PIXEL_BLACK and all 0's for
;     p == PIXEL_WHITE. This ensures the selected pixel is set to the correct
;     value, regardless of which of the four pixels it is.
;   - Sends these computed write mask and data nibbles to the VRAM
;
; Arguments (After execution of SAVE):
;   %i0 = x coordinate of desired pixel
;   %i1 = y coordinate of desired pixel
;   %i2 = p = color to paint specified pixel
;
; Return Values: None
;
; Outputs: Pixel at (x, y) painted with color p.
;          (Corresponding bit in VRAM is set)
;
; Registers Destroyed: None (Registers SAVED/RESTORED)
; Stack depth: 0
;
; Shared Variables: None
;
; Data Structures/Algorithms: None
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
plot_pixel:
    SAVE    %sp, 0 ; no stack space is required

    ; Load VRAM base address
    MOVIA   %l0, na_vram_0

    ; Compute offset from VRAM base address:
    ; Offset = (VRAM_row << 8) + VRAM_column
    ;        = (y << 8) + (x >> 2)

    LSLI    %i1, 8      ; move y coordinate into the row portion of offset
    ADD     %l0, %i1
    MOV     %l1, %i0    ; Get a copy of the x coordinate
    LSRI    %l1, 2      ; get the nibble in which the pixel appears

    ADD     %l0, %l1    ; Add in the column portion of the nibble offset

    MOVI    %l2, 3
    AND     %i0, %l2    ; get the x offset within that nibble

    BGEN    %l1, 7      ; set the leftmost pixel enable bit
    LSR     %l1, %i0    ; shift the pixel enable bit into the 'x' location
                        ; within the write mask

    ; If the color, p, is PIXEL_WHITE, set all the bits in the graphics nibble
    ; to '0' (to clear the pixel). Otherwise, set all the bits in the graphics
    ; nibble to '1' (blacken the pixel). Assuming the PIXEL_WHITE value is 0
    ; and the PIXEL_BLACK value is 0x0F, p can simbly be ORed into the data
    ; byte.
    OR      %l1, %i2

    FILL8   %r0, %l1    ; Fill all bytes of this word with the computed
                        ; graphics byte so ST8D will read it in all cases

    ST8D    [%l0], %r0

    RESTRET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; void clear_display(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Sets all the LCD pixels to white.
; Operation:
;   Iterates over the portions of VRAM mapped to pixels on the LCD, and paints
;   them all white 16 pixels (4 nibbles) at a time.
;
; Arguments: None
; Return Values: None

; Outputs: All pixels on the LCD are painted white.
;          (All bits in VRAM that are displayed on the LCD are set to '0')
;
; Registers Destroyed: None
; Stack depth: 0
; Shared Variables: None
; Data Structures/Algorithms: None
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
clear_display:
    SAVE    %sp, 0                  ; no stack space is required

    ; Load VRAM base address
    MOVIA   %l0, na_vram_0
    MOVIA   %l3, CLEAR_PATTERN      ; Load pattern to clear 4 graphics nibbles

    MOVIA   %l1, SIZE_Y             ; loop over all LCD rows
    MOVIA   %l4, ROW_ADVANCE        ; Ammount added to address to advance row
row_loop:
    MOVIA   %l2, LCD_NIBBLES_WIDE   ; loop over all the nibbles in the row

col_loop:
    ST      [%l0], %l3              ; Clear 4 nibbles (16 pixels) of the LCD

    SUBI    %l2, NIBBLES_PER_WRITE  ; Subtract nibbles cleared from remaining
    IFS     cc_nz
    BR      col_loop                ; Continue until there are none remaining
    ADDI    %l0, COLUMN_ADVANCE     ; Move to the next set of 4 nibbles

    SUBI    %l1, ROWS_PER_LOOP      ; Subtract rows cleared from rows remaining
    IFS     cc_nz
    BR      row_loop                ; Continue until there are no rows remaining
    ADD     %l0, %l4                ; Advance to the start of the next row

    RESTRET
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; File: interrupts.s
; Description:
;       Routine to enable interrupts for the NIOS CPU.
;       The only interrupts acknowledged are from the keypad, so
;       only the na_key_ready_pio's interrupt need be enabled.
; Input:        None.
; Output:       None.
;
; Known Bugs:   None
;
; Revision History:
;       09/06/08    Julian Panetta  Initial Revision
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.include "excalibur.s"

.text
.global init_interrupts

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; void init_interrupts(void)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   Enables all the interrupts to be acknowledged by the NIOS.
;   This should only be called AFTER the ISRs to handle these interrupts have
;   already been installed.
; Operation:
;   - Sets the key_ready pio's interrupt mask bit to '1' so that the debounced
;     key bit line can interrupt the NIOS' operation.
;
; Arguments: None
; Return Values: None
; Outputs: None
; Registers Destroyed: None (Registers SAVED/RESTORED)
; Stack depth: 0
; Shared Variables: None
; Data Structures/Algorithms: None
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
init_interrupts:
    SAVE    %sp, 0  ; No stack space is needed

    MOVIA   %l0,  na_key_ready_pio
    MOVI    %l1, 1
    STP [%l0, np_piointerruptmask], %l1

    RESTRET
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; File: constants.s
; Description:
;   Defines constants used by the keypad and display hardware drivers.
;   Also defines global software constants and constants related to the status
;   LED.
;
; Revision History:
;   11/17/08    Julian Panetta  Initial Revision
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Global Constants
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.equ true,  1       ; Boolean true value
.equ false, 0       ; Boolean false value

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Status LED Constants
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.equ LED_ON,    1   ; Value to write to LED PIO to turn on Status LED
.equ LED_OFF,   0   ; Value to write to LED PIO to turn off Status LED

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Display Constants
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.equ PIXEL_BLACK,       0X0F        ; Pixel color passed to draw black pixels.
.equ PIXEL_WHITE,       0X00        ; Pixel color passed to draw white pixels.
.equ CLEAR_PATTERN,     0xF0F0F0F0  ; Word value that will clear 16 bits of VRAM

; VRAM/LCD Properties
.equ SIZE_X,            640         ; X resolution of the LCD
.equ SIZE_Y,            200         ; Y resolution of the LCD
.equ LCD_NIBBLES_WIDE,  (SIZE_X / 4); Number of nibbles in VRAM per LCD row
.equ VRAM_WIDTH,        256         ; Number of nibbles in VRAM per VRAM row

; Constants used for clearing the LCD efficiently
.equ NIBBLES_PER_WRITE, 4           ; Number of nibbles cleared per write
.equ COLUMN_ADVANCE,    NIBBLES_PER_WRITE   ; Number of nibbles to skip over to
                                            ; advance to the next column
.equ ROWS_PER_LOOP,     1               ; Number of rows cleared per iteration
.equ ROW_ADVANCE,   (VRAM_WIDTH - LCD_NIBBLES_WIDE) ; Number of nibbles to skip
                                                    ; to advance to the next row


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Keypad Constants
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.equ KEY_UP,        0x02    ; <Up>
.equ KEY_DOWN,      0x22    ; <Down>
.equ KEY_LEFT,      0x11    ; <Left>
.equ KEY_RIGHT,     0x14    ; <Right>
.equ KEY_MENU,      0x12    ; <Menu>
.equ KEY_ILLEGAL,   0x00    ; Illegal Key
```

```
/****************************************************************************/
/*                                                                          */
/*                                 STUBFNCS                                 */
/*                        Oscilloscope Stub Functions                       */
/*                        Digital Oscilloscope Project                      */
/*                                 EE/CS 52                                 */
/*                                                                          */
/****************************************************************************/

/*
   This file contains stub functions for the hardware interfacing code for the
   Digital Oscilloscope project.  The file is meant to allow linking of the
   main code without necessarily having all of the low-level functions or
   hardware working.  The functions included are:
      key_available   - check if a key is available
      getkey          - get a key
      clear_display   - clear the display
      plot_pixel      - plot a pixel
      set_sample_rate - set the sample rate
      set_trigger     - set the trigger level and slope
      set_delay       - set the trigger delay
      start_sample    - start sampling
      sample_done     - sampling status

   The local functions included are:
      none

   The locally global variable definitions included are:
      none


   Revision History
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/13/94  Glen George       Changed set_sample_rate to return SIZE_X.
      5/9/06   Glen George       Updated start_sample stub to match the new
                                 specification.
      11/4/08  Julian Panetta    Altered stub functions to implement test
                                 analog sample. Also removed stub functions
                                 for keypad/display which have real
                                 implmeentations.
*/



/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"
#include  "scopedef.h"

/* Variables used to store state of analog hardware simulator */
static long int sample_rate;
static int sample_level, sample_slope;
static long int sample_delay;
static int is_sampling;

/* Buffer to store simulated analog input data */
static char sample_data[SIZE_X];

/* sampling parameter functions */

int  set_sample_rate(long int rate)
```

```c
{
    sample_rate = rate;
    return  SIZE_X;
}

void  set_trigger(int level, int slope)
{
    sample_level = level;
    sample_slope = slope;
    return;
}

void  set_delay(long int delay)
{
    sample_delay = delay;
    return;
}



/* sampling functions */

void  start_sample(int auto_trigger)
{
    get_test_sample(sample_rate, SIZE_X, sample_data);
    is_sampling = 1;
}

unsigned char far  *sample_done()
{
    /* Only return NULL once per sample */
    if (!is_sampling)
        return  NULL;
    else    {
        is_sampling = 0;
        return sample_data;
    }
}
```

```
/****************************************************************************/
/*                                                                          */
/*                                INTERFAC.H                                */
/*                           Interface Definitions                          */
/*                               Include File                               */
/*                       Digital Oscilloscope Project                       */
/*                                 EE/CS 52                                 */
/*                                                                          */
/****************************************************************************/

/*
   This file contains the constants for interfacing between the C code and
   the assembly code/hardware for the Digital Oscilloscope project.  This is
   a sample interface file to allow compilation of the .c files.


   Revision History:
      3/8/94   Glen George       Initial revision.
      3/13/94  Glen George       Updated comments.
      3/17/97  Glen George       Added constant MAX_SAMPLE_SIZE and removed
                                 KEY_UNUSED.
     11/04/08  Julian Panetta    Updated to match constants of my hardware
                                 drivers.
*/

#ifndef  __INTERFAC_H__
    #define  __INTERFAC_H__

/* library include files */
  /* none */

/* local include files */
  /* none */

/* constants */

/* keypad constants */
#define  KEY_MENU    0x12       /* <Menu>      */
#define  KEY_UP      0x02       /* <Up>        */
#define  KEY_DOWN    0x22       /* <Down>      */
#define  KEY_LEFT    0x11       /* <Left>      */
#define  KEY_RIGHT   0x14       /* <Right>     */
#define  KEY_ILLEGAL    0       /* illegal key */

/* display constants */
#define  SIZE_X         640     /* size in the x dimension */
#define  SIZE_Y         200     /* size in the y dimension */
#define  PIXEL_WHITE      0     /* pixel off */
#define  PIXEL_BLACK    0xF     /* pixel on */

/* scope parameters */
#define  MIN_DELAY         0    /* minimum trigger delay */
#define  MAX_DELAY     50000    /* maximum trigger delay */
#define  MIN_LEVEL         0    /* minimum trigger level (in mV) */
#define  MAX_LEVEL      5000    /* maximum trigger level (in mV) */

/* sampling parameters */
#define  MAX_SAMPLE_SIZE   2400 /* maximum size of a sample (in samples) */

#endif
```